Transformation of cryptographic primitives: provable security and proof presentation

Jaime Gaspar

7 January 2019 (submission)   12 November 2019 (revision)

Ph.D. dissertation

School of Computing   University of Kent

63144 words (approximately)   186 pages (exactly)

Abstract

We analyse transformations between cryptographic primitives and, for each transformation, we do two studies: its provable security (proving that if the original cryptographic primitive is secure, then the transformed cryptographic primitive is also secure); its proof presentation (exploring improved ways of presenting the proof). Our contributions divide into two sets: security proofs (sometimes new proofs and sometimes variants of known proofs); proof presentations (inspired by our security proofs) and extraction of lessons learned from them.

Acknowledgements

First of all, I would like to very, very gratefully thank my past de jure doctoral advisor, present de facto doctoral advisor and present external advisor, Eerke Boiten, for all his kind advice.

I would like to gratefully thank my past supervisory team member and present de jure doctoral advisor, Simon Thompson, for all his kind advice.

I would like to gratefully thank my present supervisory team member, Julio Hernandez-Castro, for all his kind advice.

I would like to gratefully thank Carlos Pérez-Delgado for all his kind advice.

I would like to gratefully thank David Galindo for being the external examiner in my viva voce.

I would like to gratefully thank Stefan Kahrs for being the internal examiner in my viva voce.

I would also like to gratefully thank the University of Kent in general and its School of Computing in particular for all their kind logistical support.

Last but not least, I would like to gratefully thank the Research Postgraduate Scholarship from the Engineering and Physical Sciences Research Council / School of Computing, University of Kent for all its kind financial support.

Finally, I would like to gratefully thank everyone mentioned for the extra time given to me to complete this Ph.D. dissertation.

Almost all content in chapter 6 is heavily based on joint work with Eerke Boiten (Gaspar and Boiten 2014) and for evaluation purposes we stipulate that that work is split in equal parts between Eerke Boiten and Jaime Gaspar.

The approximated word count on the cover was calculated under Linux with `cat *.tex *.bib | wc -w`.

Contents

Part I

Introduction

Chapter 1

Introduction

1.1 Introduction

1.1. Roughly speaking, our subject of study is proofs in cryptography (in the intersection of mathematics and computer science); in more detail, it is

1. transformation of cryptographic primitives;

from the points of view of

2. provable security;

3. proof presentation.

We start by introducing, motivating and justifying this subject.

1.2. In this chapter, we give only a contribution modestly worth of notice: the definitions, propositions and proofs around the toy cipher.

1.2 Problem

1.3. Definitions, theorems and proofs in cryptography in general and provable security (introduced in section 1.4 below) in particular are often complicated because they may involve more than one of the following five theories:

1. computability theory to model a cryptographic primitive (or protocol) and an adversary or breaker both with limited computational power;

2. probability theory to model the success likelihood and the guessing strategy of an adversary or breaker;

3. asymptotic theory to give meaning to negligible success probabilities as a function of the size of a security parameter (such as a key);

4. number theory (or other theories such as lattice theory) to provide hard problems that are used to construct hard-to-break cryptographic primitives;

5. cryptographic theory with its rich "zoo" of

(a) cryptographic primitives and protocols;

(b) security notions;

(c) proof methods;

(d) known results and assumed conjectures;

(e) disproved bad ideas;

(f) computer-assisted tools;

and so on. (Although we focus on cryptographic primitives, perhaps we should remark that cryptographic protocols, for example the framework of universal composability (Ristenpart, Shacham and Shrimpton 2011, section 1), also face some challenges.)

1.4. The complexity of definitions, theorems and proofs in cryptography in general and provable security in particular increases the likelihood that mistakes "slip through undetected", thus somewhat undermining the point of having definitions, theorems and proofs to achieve a mathematical degree of certainty. This was perhaps first clearly articulated by Shai Halevi when he pointed out that the cryptographic community produces proofs in larger numbers, faster and with greater complexity than it can carefully verify, so some incorrect "proofs" are bound to go unnoticed (Halevi 2005, page 2). The evaluation of how serious this problem is varies:

1. some, such as Neal Koblitz et al., have considered that the problem is so large that it amounts to a crisis;

2. others, such as Oded Goldreich et al., have argued that it is an exaggeration to consider the problem a crisis.

In the next two paragraphs we summarise their points of view.

1.5. Neal Koblitz, mostly with Alfred J. Menezes, wrote a series of articles (often with titles starting with "Another look at") in which they presented some criticism of provable security. We would say that their main criticisms are the following.

1. Security proofs for asymmetric cryptosystems often focus only on the non-invertibility of the one-way function which the cryptosystem is based on while ignoring that most attacks actually exploit not the function but the cryptographic protocol wrapped around the cryptosystem (an example being Daniel Bleichenbacher's attack on the Rivest-Shamir-Adleman cipher RSA (Bleichenbacher 1998)) (Koblitz and Menezes 2007, page 4).

2. A security proof guarantees resistance against attacks included in the underlying security notion but not against other attacks (two examples being side-channel attacks (Koblitz and Menezes 2007, page 8), and that Shafi Goldwasser, Silvio Micali and Ronald Rivest's most-followed definition of a secure signature scheme does not include the duplicate signature key selection attack (Koblitz and Menezes 2013, page 4)) (Koblitz and Menezes 2007, page 8) (Koblitz and Menezes 2013, page 4).

3. Sometimes we should aim for more than “formalistic” proofs by reduction of theoretical interest, namely to aim for “tight” reductions of a more applied interest giving meaningful estimations for security parameters' sizes (an example being that whether RSA and the Probabilistic Signature Scheme PSS are considered equally secure may depend on what kind of proof we aim for (Koblitz and Menezes 2007, page 17)) (Koblitz and Menezes 2007, pages 17 and 29) (Koblitz and Menezes 2006, sections 4 and 6).

4. Some security proofs, due to their complexity and delicateness, may contain subtle mistakes that remain undetected for some time, casting a shadow over the degree of trust that we may put on proofs (two examples being the mistake found by Victor Shoup (Shoup 2002, section 4) in Mihir Bellare and Phillip Rogaway's security proof of the RSA Optimal Asymmetric Encryption Padding RSA-OAEP and the mistake found by David Galindo (Galindo 2005, section 3.2) in Dan Boneh and Matthew Franklin's security proof of their identity based encryption) (Koblitz and Menezes 2007, section 4.2).

5. Some security proofs, due to their technicality and impenetrability, may be inaccessible and therefore unconvincing for more applied and less theoretical people, reducing the impact of those proofs (Koblitz and Menezes 2007, sections 4.4 and 8).

6. Security proofs would be more credible if there were in cryptography a tradition of careful review of proofs without the pressure of conference deadlines, akin to the tradition in mathematics (an example being Mihir Bellare and Phillip Rogaway's less reviewed proof of the security of RSA-OAEP (Bellare and Rogaway 1995, appendix A) versus Andrew Wiles' more carefully reviewed proof of Fermat's last theorem) (Koblitz and Menezes 2007, section 4.4) (Koblitz 2007, page 976).

7. Minor changes in a cryptosystem may break security proofs (an example being that the Digital Signature Algorithm DSA broke the security proof of the Schnorr signature (Koblitz and Menezes 2007, page 26)) (Koblitz and Menezes 2007, page 26).

8. If the parts of a hybrid cryptosystem are not independent, then security proofs under the random oracle model (for idealised hash functions) of that cryptosystem may not imply real security (with real hash functions) (an example being Mihir Bellare, Alexandra Boldyreva, Adriana Palacio's hybrid hashed ElGamal key-encapsulation composed with symmetric encryption (Bellare, Boldyreva and Palacio 2004, proposition 2)) (Koblitz and Menezes 2007, page 30).

9. A provable-security result may suggest one thing and another provable-security result may suggest the opposite (an example being Dan Boneh and Venkatesan Ramarathnam's result suggesting that breaking RSA may be easier than factoring (Boneh and Venkatesan 1998, theorems 3.3 and 3.7), and Daniel R. L. Brown's result suggesting the opposite (Brown 2016, theorem 9)) (Koblitz and Menezes 2006, section 3.1).

10. Despite provable security's goal of giving precise definitions of security, some definitions may still contain some ambiguity (an example being that the definition of an asymmetric cipher being resistant against chosen-ciphertext attacks has four possible readings of which at least three are non-equivalent (Bellare, Hofheinz and Kiltz 2015, figures 1–2)) (Koblitz and Menezes 2013, section 3.1).

11. Focusing on achieving provable security for a certain security definition may lead to a simplification of a cryptosystem by removing security features as long as security remains provable, but the removal of features may:

    (a) weaken the cryptosystem against attacks not included in the security definition;

    (b) remove features useful when the security proof is mistaken;

    (c) remove features useful when the underlying security model is unrealistic;

    (an example being Hugo Krawczyks' hashed-variant simplification HMQV of the Menezes-Qu-Vanstone authenticated key-agreement protocol MQV, which permitted new attacks and had a mistake in the security proof (Menezes 2007, section 3)) (Koblitz and Menezes 2006, section 5).

12. Sometimes the security proof of a cryptosystem relies on the hardness of a problem that instead of being natural is contrived (an example being the security of the Boneh-Boyen signature relying on the strong Diffie-Hellman problem (Boneh and Boyen 2008, sections 3.1–3.2)) (Koblitz and Menezes 2010, section "The Strong Diffie-Hellman Problem").

1.6. Oded Goldreich and others counter-criticised Neal Koblitz's criticism. We would say that their main criticisms are the following.

1. Oded Goldreich reads Neal Koblitz's criticism as close to a rejection of rigorous analysis (in favour of informality and intuition) and advocates that rigorous analysis (not informality nor intuition) is the right methodology for a scientific cryptography (Goldreich, Barak, Katz, Krawczyk and Koblitz 2007).

2. Boaz Barak argues that the situation in cryptography is not worse than in mathematics in general (an example being that mistakes in proofs occur in all mathematics) (Goldreich, Barak, Katz, Krawczyk and Koblitz 2007).

3. Jonathan Katz finds that Neal Koblitz's criticism is in part a fallacious *ad hominem* argument (an example being the suggestion that some cryptographers publish papers of reduced originality or with small improvements of previous work) (Goldreich, Barak, Katz, Krawczyk and Koblitz 2007).

4. Hugo Krawczyk argues that provable security does play an important role in the design and analysis of real-world cryptosystems (an example being that HMQV brings improvements to MQV guided by provable security) (Goldreich, Barak, Katz, Krawczyk and Koblitz 2007).

5. Avi Wigderson argues that cryptographic scenarios are adversarial and unexpected, so empirical testing of cryptosystems is less reliable, thus we have to rely more on mathematical proofs, which increases the importance of provable security (Wigderson 2008).

6. Ivan Damgård remarks that non-"tight" security proofs may not give estimations for the size of security parameters but nevertheless they do prove that a cryptosystem is secure (Damgård 2007, section 2.1).

1.7. To illustrate the fact that sometimes "proofs" in cryptography have mistakes, let us give an example of a "proof" in cryptography that was published and afterwards discovered to have a mistake: the security "proof" (Ristenpart and Rogaway 2007, sections 6–7) (Ristenpart and Rogaway 2015, sections 6–7) of Thomas Ristenpart and Phillip Rogaway's eXtension by Latin Squares XLS (Ristenpart and Rogaway 2007, section 3) (Ristenpart and Rogaway 2015, section 3) and its mistake discovered by Mridul Nandi (Nandi 2014, section 3) (Nandi 2015, appendix A).

Notions and notations Before we proceed to the example, we need to introduce some notions and notations that we will use in it:

1. if $L \subseteq \mathbb{N}$, then let $\{0,1\}^L := \bigcup_{l \in L} \{0,1\}^l$;
2. let $\{0,1\}^{2*} := \bigcup_{n \in \mathbb{N}} \{0,1\}^{2n}$;
3. if $n \in \mathbb{N}$, then let $[0 \mathinner{..} n] := \{0, 1, 2, \dots, n\}$;
4. if $X, Y \subseteq \mathbb{N}$, then let $X + Y := \{x + y \mid x \in X,\ y \in Y\}$;
5. if $x, y \in \{0,1\}^*$, then let $x \| y$ or xy denote the concatenation of x and y;
6. if $x \in \{0,1\}^{2*}$, then let $x_{\leftarrow}$ and $x_{\rightarrow}$ denote respectively the left and right half of x;
7. if $x \in \{0,1\}^*$ and $k \in \mathbb{N}$, then let $x \lll k$ and $x \ggg k$ denote respectively the left and right circular shift of x by k bits (for example $x_1 x_2 x_3 \dots x_n \lll 2 = x_3 x_4 x_5 \dots x_n x_1 x_2$, where $x_1, x_2, x_3 \dots, x_n \in \{0,1\}$);
8. if $b \in \{0,1\}$, then let $\bar{b} := b \oplus 1$ denote the bit obtained by flipping the bit b;
9. if $n \in \mathbb{N} \setminus \{0\}$, then let U_n denote the uniform random variable over $\{0,1\}^n$;
10. if $E \colon \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ is an encryption function (which encrypts, with a key $k \in \mathcal{K}$, a plaintext $p \in \mathcal{P}$ to a ciphertext $c = E(k,p) \in \mathcal{C}$), then we say that E is length-preserving if and only if $\forall k \in \mathcal{K}\ \forall p \in \mathcal{P}\ |E(k,p)| = |p|$;
11. if $E \colon \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ and $E' \colon (\mathcal{K} \times \mathcal{K}') \times \mathcal{P}' \to \mathcal{C}'$ are encryption functions with $\mathcal{P} \subseteq \mathcal{P}'$, then we say that E' extends E if and only if $\forall k \in \mathcal{K}\ \forall k' \in \mathcal{K}'\ \forall p \in \mathcal{P}\ E'((k,k'),p) = E(k,p)$;
12. if $E \colon \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ (where $\mathcal{C} := \mathcal{P}$) is an encryption function with associated decryption function $D \colon \mathcal{K} \times \mathcal{C} \to \mathcal{P}$ (that is $\forall k \in \mathcal{K}\ \forall p \in \mathcal{P}\ D(k, E(k,p)) = p$), then we say that E is a pseudorandom permutation (Katz and Lindell 2015, definition 3.28) if and only if informally

E is length preserving and for all polynomial-time probabilistic algorithms $A^{O,O'}$ (called distinguishers) with access to a pair (O, O') of oracles, the "probability"

$$\Pr[A^{E(k,\cdot),\,D(k,\cdot)}(1^n) = 1] - \Pr[A^{\pi,\pi^{-1}}(1^n) = 1]$$

that $A^{O,O'}$ can distinguish (in the sense of giving different outputs) between $(O, O') = (E(k,\cdot), D(k,\cdot))$ (for a "uniformly random" $k \in \mathcal{K}$) and $(O, O') = (\pi, \pi^{-1})$ (for a "uniformly random" and length-preserving permutation π of $\mathcal{P}$) is "negligible" (in the sense that it vanishes super-polynomially in $|k|$).

Motivation To facilitate their definition and the study of their security, encryption functions are sometimes defined to encrypt not all binary strings (the full $\{0,1\}^*$) but only some binary strings (for example $\{0,1\}^n$). This raises the question of how to extend an encryption function E to an encryption function E'' that encrypts more binary strings. Naturally, we want that the extension preserves properties of E (for example being length-preserving) and the security of E (for example being a pseudorandom permutation). Thomas Ristenpart and Phillip Rogaway proposed the construction XLS that extends E to more strings (but not all) while preserving its property (being length preserving) and its security (being a pseudorandom function) but at the expense of an auxiliary encryption function E'.

Construction of XLS Let $n' \in \mathbb{N} \setminus \{0\}$. We assume that we have

1. a main length-preserving encryption function

$$E\colon \{0,1\}^K \times \{0,1\}^N \to \{0,1\}^N$$

(where $\emptyset \neq K \subseteq \mathbb{N}$ and $\emptyset \neq N \subseteq \mathbb{N} \setminus [0\,..\,n'-1]$);

2. an auxiliary encryption function

$$E'\colon \{0,1\}^{K'} \times \{0,1\}^{n'} \to \{0,1\}^{n'}$$

(where $\emptyset \neq K' \subseteq \mathbb{N}$);

we define

3. the mixing function

$$\begin{aligned} \mathrm{m}\colon \{0,1\}^{2*} &\to \{0,1\}^{2*} \\ x &\mapsto \big(x_{\leftarrow} \oplus ((x_{\leftarrow} \oplus x_{\rightarrow}) \lll 1)\big) \,\|\, \big(x_{\rightarrow} \oplus ((x_{\leftarrow} \oplus x_{\rightarrow}) \lll 1)\big) \end{aligned};$$

and then we construct

4. the length-preserving encryption function

$$E''\colon \mathcal{K}'' \times \{0,1\}^{N''} \to \{0,1\}^{N''}$$

(where $\mathcal{K}'' := \{0,1\}^K \times \{0,1\}^{K'}$ and $N'' := N + [0\,..\,n'-1]$) that extends E and is defined by

(a) if $p \in \{0,1\}^N$, then $E''((k,k'),p) := E(k,p)$;

(b) if $p \notin \{0,1\}^N$, then $E''((k,k'),p) := c$ where c is the last value of c when we apply the algorithm described in the following table (where $l := \max\{l \in N \mid \exists l' \in [0 \mathinner{..} n'-1]\ |p| = l + l'\}$, $l' := |p| - l$ and j ranges over $[0 \mathinner{..} i-1]$).

1	Set c to be equal to p.	
2	Divide c into blocks of length n' with the last block being incomplete and possibly empty.	$c = \boxed{x_1}\boxed{...}\boxed{x_{i-2}}\boxed{x_{i-1}}\boxed{x_i} \quad \lvert x_j \rvert = n' > \lvert x_i \rvert$
	Encrypt the penultimate block with E'.	$c = \boxed{x_1}\boxed{...}\boxed{x_{i-2}}\boxed{E'(k', x_{i-1})}\boxed{x_i}$
3	Divide c into three blocks, the second one of length 1 and the third one of length $2l'$.	$c = \boxed{y_1}\boxed{y_2}\boxed{y_3} \quad \lvert y_2 \rvert = 1 \quad \lvert y_3 \rvert = 2l'$
	Flip the bit in the second block and mix the third block with m.	$c = \boxed{y_1}\boxed{\overline{y_2}}\boxed{\mathrm{m}(y_3)}$
4	Divide c into two blocks, the first one of length l.	$c = \boxed{z_1}\boxed{z_2} \quad \lvert z_1 \rvert = l$
	Encrypt the first block with E.	$c = \boxed{E(k, z_1)}\boxed{z_2}$
5	Repeat step 3.	$c = \boxed{y'_1}\boxed{y'_2}\boxed{y'_3} \quad \lvert y'_2 \rvert = 1 \quad \lvert y'_3 \rvert = 2l'$ $c = \boxed{y'_1}\boxed{\overline{y'_2}}\boxed{\mathrm{m}(y'_3)}$
6	Repeat step 2.	$c = \boxed{x'_1}\boxed{...}\boxed{x'_{i-2}}\boxed{x'_{i-1}}\boxed{x'_i} \quad \lvert x'_j \rvert = n' > \lvert x'_i \rvert$ $c = \boxed{x'_1}\boxed{...}\boxed{x'_{i-2}}\boxed{E'(k', x'_{i-1})}\boxed{x'_i}$
7	Output c.	

Decryption D'' is done essentially as encryption E'' but inverting both the order of the steps and the functions $E(k,\cdot)$ and $E'(k',\cdot)$ (that is replacing those functions by their corresponding decryptions $D(k,\cdot)$ and $D'(k',\cdot)$) (there is no need to invert m because $\mathrm{m}^{-1} = \mathrm{m}$).

Security The security claim for XLS is essentially the following: if E and E' are pseudorandom permutations, then E'' is a pseudorandom permutation (Ristenpart and Rogaway 2007, theorem 2) (Ristenpart and Rogaway 2015, theorem 3).

Counterexample Mridul Nandi showed that the security claim is false (assuming the existence of pseudorandom permutations) by proving that if $E = E'$, then the algorithm $A^{O,O'}$ that, when given oracle access to $(O, O') = \big(E((k,k'),\cdot),$

$D((k, k'), \cdot)\big)$, computes

$$\begin{aligned} p &:= U_{2n'-1}, \\ c &:= E''((k, k'), p), \\ p' &:= D''((k, k'), c_{\leftarrow}\overline{c_{\rightarrow}}), \\ p'' &:= p_{\rightarrow} \oplus \overline{p'_{\rightarrow}} \oplus ((p_{\rightarrow} \oplus \overline{p'_{\rightarrow}}) \gg 2), \\ c' &:= E''((k, k'), p'_{\leftarrow} \| (p_{\rightarrow} \oplus p'')) \end{aligned}$$

and outputs the truth value (0 or 1) of $c'_{\rightarrow} = p'' \oplus c_{\rightarrow}$, is a distinguisher for E'' with non-negligible probability of at least $1/2 - 2^{1-n'}$ (Nandi 2014, theorem 2) (Nandi 2015, section 2.3).

Mistake After presenting his counterexample, Mridul Nandi identified the mistake in the security proof (Nandi 2015, appendix A). We present here our take on the mistake, which is somewhat simpler and has the advantage of not requiring familiarity with the "semantics" of the proof because it is almost entirely "formal". Thomas Ristenpart and Phillip Rogaway have a pseudocode for a game G4 dealing with some variables M^i_k and that in part says

1. let $i \in [1 \mathinner{..} j]$ be such that $M^i_2 = M^j_2$;
2. (a) if $i < j$, then let $M^j_4 := M^i_4$ and $M^j_5 := M^i_5$;
   (b) otherwise let M^j_4 and M^j_5 be uniformly random and independent (Ristenpart and Rogaway 2007, page 18);

moreover, they are considering the case

3. $M^i_2 \neq M^j_2$;

and they claim that

4. M^i_4, M^i_5, M^j_4 and M^j_5 are independent (Ristenpart and Rogaway 2007, page 19).

The mistake is that point 4 is false: from point 3 we get $i \neq j$, so from point 1 we get $i < j$, thus the point that applies is point 2a and not point 2b, hence we have $M^j_4 := M^i_4$ and $M^j_5 := M^i_5$, therefore we do not have point 4. Thomas Ristenpart and Phillip Rogaway acknowledged the mistake identified by Mridul Nandi and retracted their article (Ristenpart and Rogaway 2015, cover page).

Solution? It is difficult to say how Thomas Ristenpart and Phillip Rogaway could have avoided the mistake. They appear to have done everything right:

1. they gave informal (in the form of words) and formal (in the form of pseudocode) descriptions to play both in the field of intuition and in the field of rigour;
2. they organised (with visual cues) the "proof" into cases (in the form of claims), subcases (indicated by a triangle $\triangleright$) and sub-subcases (indicated by a bullet $\bullet$) to help keep track of where one is in the "proof";

3. they used notational conventions fairly easy to understand (for example plaintexts involve the letter M as in M^j, and the parsing of the binary string M^j as the concatenation of the binary strings M_1^j, M_2^j and M_3^j with respectively lengths m, n and s is denoted by "$M_1^j\, M_2^j\, M_3^j \leftarrow M^j$ *of lengths* m, n, s");

and so on. Despite all this care, a mistake still slipped through. It is hard to say what could have been done better. The only answer that occurs to us is that the

1. mixing of objects in different levels (for example bits are in a lower level than binary strings, which in turn are in a lower level than functions mapping binary strings to binary strings, and so on);
2. going up and down, by using cases, subcases and sub-subcases, in a somewhat complicated tree structure underlying the "proof";

may suggest that it would be neater to divide the "proof" into layers of abstraction (for example first we work only with the lower level, then we abstract from it and we work only at the second level, then we abstract from it again and work only at the third level, and so on), but this is easier said than done.

1.8. Independently of whether the problem with definitions, theorems and proofs in cryptography in general and provable security in particular amounts to a crisis or not, we believe that the problem warrants effort to find solutions. We would say that a possible solution for the problems has the following four components:

1. transformation of cryptographic primitives;
2. provable security;
3. proof presentation;
4. automated/interactive theorem provers.

In the next four sections we explain these four components in more detail. To illustrate this discussion with examples, we first need to introduce the notions of

1. toy cipher;
2. toy one-way function;

that we will use as simple running examples through those sections.

1.9. The following definitions are about toy ciphers.

1. Let $m, n \in \mathbb{N}$ be fixed parameters.
2. A *toy pseudorandom generator* is a function $G\colon \{0,1\}^m \to \{0,1\}^n$ whose inputs $s \in \{0,1\}^m$ are called *seeds* and whose outputs $G(s) \in \{0,1\}^n$ are called (prefixes of) *streams* (informally, we hope that a toy pseudorandom generator expands a truly-random seed into a not-truly-random-but-random-enough stream, although this is not actually required by the definition).

3. A toy pseudorandom generator G is *toy secure* if and only if it is not a constant function (informally, we hope that the outputs of G look random, so, with high probability, they should not be all equal, and we take this latter condition as our notion of security; it is admittedly a weak condition but it turns out to be good enough for our purposes).

4. A *toy cipher* is an ordered pair $C = (E, D)$ such that:

   (a) $E\colon \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ is a function called *encryption* whose inputs $k \in \{0,1\}^m$ and $p \in \{0,1\}^n$ are respectively called *keys* and *plaintexts* (informally, the encryption codes a plaintext using a secret key);

   (b) $D\colon \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n$ is a function called *decryption* whose inputs $k \in \{0,1\}^m$ and $c \in \{0,1\}^n$ are respectively called *keys* and *ciphertexts* (informally, the decryption decodes a ciphertext using the same secret key);

   (c) $\forall k \in \{0,1\}^m\ \forall p \in \{0,1\}^n\ D(k, E(k,p)) = p$ is a property that holds true and is called *correctness* (informally, correctness means that the decryption undoes the encryption).

5. The *toy stream cipher* $C_G = (E_G, D_G)$ induced by a toy pseudorandom generator G is the toy cipher defined by:

   (a) $E_G(k,p) := G(k) \oplus p$;

   (b) $D_G(k,c) := G(k) \oplus c$;

   (informally, a toy stream cipher encrypts by mixing the stream with the plaintext through xor, decrypts by mixing the same stream with the ciphertext, and enjoys correctness because the mixings cancel out in the sense of $x \oplus x \oplus y = y$).

6. A *toy breaker* for a toy cipher $C = (E, D)$ is a function $B\colon \{0,1\}^n \to \{0,1\}^n$ such that $\forall k \in \{0,1\}^m\ \forall p \in \{0,1\}^n\ B(E(k,p)) = p$ (informally, a toy breaker decrypts any ciphertext without the key akin to Dan Brown's TRANSLTR (Brown 1998, chapter 4) but specialised to break C). A toy cipher is *toy secure* if and only if it has no toy breakers (informally, meaning that there is no way to bypass the key).

The following definitions are about toy one-way functions.

1. Let $\underline{0}$ denote the identically-zero function $\underline{0}\colon \mathbb{N} \to \mathbb{R}$ defined by $\underline{0}(n) := 0$.

2. Let $\mathcal{N}$ be a set of functions $f\colon \mathbb{N} \to \mathbb{R}$ such that $\underline{0} \in \mathcal{N}$ and let us say that a function $f\colon \mathbb{N} \to \mathbb{R}$ is *toy negligible* if and only if $f \in \mathcal{N}$ (informally, we hope that a toy negligible function is a function that vanishes fast at infinity, for example such that $\forall k \in \mathbb{N}\ \lim_{n\to+\infty} n^k f(n) = 0$, although this is not actually required by the definition). We often write, say, $2^{-n} \in \mathcal{N}$, when we actually mean $f \in \mathcal{N}$ where $f\colon \mathbb{N} \to \mathbb{R}$ is defined by $f(n) := 2^{-n}$, or in other words, $\lambda n\,.\,2^{-n} \in \mathcal{N}$ where n ranges over $\mathbb{N}$.

3. Let us say that two functions $f, g\colon \mathbb{N} \to \mathbb{R}$ are *toy identical*, and write $f \approx g$, if and only if $|f - g| \in \mathcal{N}$ (later on it will be of particular interest that if f gives the probability $f(n) = \Pr P(n) \geq 0$ of some predicate P, then $f \approx \underline{0} \iff f \in \mathcal{N}$).

4. Let $x \xleftarrow{\$} \{0,1\}^n$ mean that x is chosen uniformly in $\{0,1\}^n$ (such an x is also often denoted U_n). Let U_n denote a random uniform variable in $\{0,1\}^n$.

5. We say that a function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *toy one-way* if and only if:

   (a) f is polynomial-time computable (informally, f, that is $x \mapsto f(x)$, is easy to compute);

   (b) there is no polynomial-time probabilistic algorithm A such that

   $$\Pr\big[f\big(A(f(U_n))\big) = f(U_n)\big] \in \mathcal{N},$$

   or in other words,

   $$\lambda n \,.\, \Pr_{x \xleftarrow{\$} \{0,1\}^n}\big[f\big(A(f(x))\big) = f(x)\big] \in \mathcal{N},$$

   where n ranges over $\mathbb{N}$ (informally, f is hard to invert, that $f(x) \mapsto x$, or more precisely, $f(x) \mapsto x'$ with $f(x) = f(x')$, is hard to compute).

1.3 Solution component: transformation of cryptographic primitives

1.10. The first component of the solution is the transformation of cryptographic primitives (or protocols). A transformation T of a first cryptographic primitive P_1 into a second cryptographic primitive P_2 is informally a construction $P_1 \overset{T}{\rightsquigarrow} P_2$ of P_2 using P_1 as a "black box".

1.11. Let us give a first example of a transformation of cryptographic primitives. The toy stream cipher C_G induced by the toy pseudorandom generator G is obtained by the transformation $G \rightsquigarrow C_G$ of the first cryptographic primitive G into the second cryptographic primitive C_G.

1.12. Let us give a second example of a transformation of cryptographic primitives. The example is a mechanical machine called Tunny (more precisely, the Lorentz cipher implemented in Tunny) used during the World War II by the German Army (Good, Michie and Timms 1945, section 11.A.c). Tunny was what we called a toy stream cipher with $m := 5 =: n$ (the 26 letters of the alphabet, keys, streams, plaintexts and ciphertexts were encoded as five-bit blocks) (Good, Michie and Timms 1945, section 11.A.a), so in particular it is obtained by the transformation $G \rightsquigarrow C_G$ of a toy pseudorandom generator G into the toy stream cipher C_G (Good, Michie and Timms 1945, section 11.B.b). Tunny had twelve wheels:

1. five χ-wheels χ_1, χ_2, χ_3, χ_4 and χ_5;

2. five ψ-wheels ψ_1, ψ_2, ψ_3, ψ_4 and ψ_5;
3. two μ-wheels μ_{61} and μ_{37} (Good, Michie and Timms 1945, section 11.B.c).

All wheels had pins around them and at each time one pin per wheel was in an active position denoting:

1. the bit 0 if the pin was lowered;
2. the bit 1 if the pin was raised;

(more precisely, the raised and lowered pins were respectively vertical and oblique cams) (Good, Michie and Timms 1945, section 11.B.j). Tunny operated as follows.

1. Tunny's toy pseudorandom generator G:
   (a) was set with a seed/key k consisting of the initial position of all the wheels (Good, Michie and Timms 1945, section 11.d.a);
   (b) generated a five-bit stream letter $G(k) := \chi_1\chi_2\chi_3\chi_4\chi_5 \oplus \psi_1\psi_2\psi_3\psi_4\psi_5$, then the χ- and ψ-wheels rotated in a way affected by the μ-wheels and Tunny was ready to generate the next stream letter (Good, Michie and Timms 1945, sections 11.B.d–f).
2. Tunny's toy stream cipher $C_G = (E_G, D_G)$ operated as by the definition of the toy stream cipher induced by G:
   (a) encrypted a plaintext letter p as $E_G(k,p) := G(k) \oplus p$;
   (b) decrypted a ciphertext letter c as $D_G(k,c) := G(k) \oplus c$ (Good, Michie and Timms 1945, section 11.B.b).

1.13. Transformation of cryptographic primitives helps solve the problem (from section 1.2) in the following three ways.

1. A transformation $P_1 \overset{T}{\rightsquigarrow} P_2$ may allow us to decompose a security proof of P_2 into two hopefully simpler security proofs:
   (a) a security proof of P_1;
   (b) a security proof of T showing that security is preserved by T, that is if P_1 is secure, then P_2 is secure.
2. A transformation $P_1 \overset{T}{\rightsquigarrow} P_2$ may allow us to abstract (step away) from some of the theories supporting cryptography, for example a security proof of P_2 may be complicated because it uses both number theory and complexity theory but the security proofs of P_1 and T may be easier because
   (a) the security proof of P_1 only uses number theory;
   (b) the security proof of T only uses computation theory.
3. A transformation $P_1 \overset{T}{\rightsquigarrow} P_2$ may allow us to increase the level of abstraction of the construction and security proof of P_2 by identifying and operating on two higher level concepts, namely P_1 and T.

(Three interesting bonuses are that:

1. T gives us a way of constructing a P_2 from an already constructed P_1;

2. we can "drop-in replacements", that is to replace

   (a) P_1 by any cryptographic primitive P_1' still secure;

   (b) T by any transformation T' still secure;

   obtaining another cryptographic primitive P_2';

3. to produced a new cryptographic primitive P_2', it suffices to change only P_1 to P_1' or only T to T', instead of having to change both P_1 to P_1' and T to T', so half of the work suffices.)

1.14. We give two examples of major texts dealing considerably with transformation of cryptographic primitives.

1. Oded Goldreich wrote a two-volume book on the foundations of theoretical cryptography (Goldreich 2004, Goldreich 2011). This book is somewhat technical, for example often:

   (a) definitions are expressed by formulas (akin to $f\big(A(f(U_n), 1^n)\big) = f(U_n)$);

   (b) proofs are formal and detailed (longer than a page);

   (c) notions have variants (what kind of computational model describes the adversary, whether probabilities are negligible or non-noticeable, including or excluding extra assumptions on lengths such as being regular or preserved, and so on).

   We would say that this book works well as a reference and research text.

2. Jonathan Katz and Yehuda Lindell wrote a book introducing theoretical cryptography (Katz and Lindell 2015). This book is less technical, for example often:

   (a) definitions are expressed by games (akin to "Alice picks an $x \leftarrow U_n$ for which she calculates $f(x)$ and gives $(f(x), 1^n)$ to Bob, then Bob calculates an $x' \leftarrow B(f(x), 1^n)$ and gives it to Alice, and Bob wins if $f(x) = f(x')$");

   (b) proofs are less formal and detailed (no longer than a page);

   (c) notions do not have variants.

   We would say that this book works well as an introductory and pedagogical text.

1.4 Solution component: provable security

1.15. Provable security is an area in cryptography that aims to guarantee the security of cryptographic primitives (and protocols) by means of mathematical proofs instead of empirical testing and analysis. To achieve this, all objects involved in the security theorem have to be mathematically defined to render them mathematically treatable, namely:

1. the cryptographic primitives;
2. the security notion;
3. the statement of the theorem;
4. the security proof of the theorem;

(and other objects that may be involved such as

1. assumptions made but not captured in the definitions, for example $\mathrm{P} \neq \mathrm{NP}$;
2. models where the proof may take place, for example the random oracle model;
3. hard problems that may underlie the cryptographic objects, for example the problem of factorising large integers).

1.16. Let us give a first example of provable security. The example consists of four parts:

1. the definitions of the cryptographic primitives
   (a) toy pseudorandom generator G;
   (b) toy stream cipher C_G induced by G;

   given in paragraph 1.9;
2. the definitions of toy security of
   (a) G;
   (b) C_G;

   given in paragraph 1.9;
3. the security theorem below;
4. the security proof below.

Theorem. If G is toy secure, then C_G is toy secure. □

Proof. Let us assume that G is toy secure, that is G is not constant, so there are $x, x' \in \{0,1\}^m$, necessarily distinct, such that their images $y := G(x)$ and $y' := G(x')$ are distinct: $(*)$ $x \neq x' \wedge G(x) = y \neq y' = G(x')$. Aiming at a contradiction, let us assume that $C_G = (E_G, D_G)$ has a toy breaker B, so $(\dagger)$ $\forall k \in \{0,1\}^m$ $\forall p \in \{0,1\}^n$ $B(E_G(k,p)) = p$.

1. Taking $k := x$ and $p := y$ in (†), for which $G(k) = y$ and so $E_G(k, p) = y \oplus y = 0^n$, we get (‡) $B(0^n) = y$.

2. Taking $k := x'$ and $p := y'$ in (†), for which $G(k) = y'$ and so $E_G(k, p) = y' \oplus y' = 0^n$, we get (‡′) $B(0^n) = y'$.

From (‡) and (‡′) we get $y = y'$, contradicting (∗). □

1.17. Let us give a second example of provable security. This example consists in proving that the toy one-wayness of an injective function $f\colon \{0,1\}^* \to \{0,1\}^*$ is preserved under self-composition $f \circ f$, and is written in a style somewhat mimicking Oded Goldreich's style (which is lighter in terms of formulas but less detailed than our style).

Theorem. If $f\colon \{0,1\}^* \to \{0,1\}^*$ is an injective toy one-way function, then $g := f \circ f$ is a toy one-way function. □

Proof. Let us prove by reducibility the hardness-to-invert of g assuming the same for f, that is given a polynomial-time probabilistic algorithm B inverting g we construct a polynomial-time probabilistic algorithm A inverting f: on input $f(x)$, A calls B to compute and output $x' := B\big(f(f(x))\big) = B(g(x))$; clearly A is a polynomial-time probabilistic algorithm. We claim that x' is an inverse of $f(x)$: since B inverts g, x' is an inverse of $g(x)$, that is $g(x) = g(x')$, or in other words, $f(f(x)) = f(f(x'))$, thus by the injectivity of f, $f(x) = f(x')$, that is x' is an inverse of $f(x)$. □

1.18. Provable security helps solve the problem (from section 1.2) in the following three ways.

1. Provable security gives more guarantees that a cryptographic primitive is secure (because provable security relies on the high standard of mathematical proofs) than empirical testing and analysis.

2. Provable security makes the security of a cryptographic primitive amenable to formal verification by a computer-assisted proof checker (because, in principle, mathematical definitions and proofs are expressible in a formal language and logic that can be mechanically verified).

3. Provable security helps to find mistakes in proofs (because this is the dual of checking their correctness).

1.5 Solution component: proof presentation

1.19. Proof presentation is a procedure in mathematics that aims to improve proofs by rewriting them in ways that increase their positive qualities such as:

1. to convince of the truth of the theorem;

2. to explain why the proof is the way it is;

3. to introduce a proof method;

4. to teach by example;

5. to help discover other proofs.

There is some dispute about whether proof presentation is a "science" (a rigorous knowledge that can be applied with little resort to creativity, akin to the algebraic or arithmetic laws and how they are used to solve equations) or an "art" (an unrigorous knowledge or one that requires considerable creativity to the applied, akin to guiding principles to write a good novel). We take a middle position here:

1. some aspects are "science"-like, for example guidelines to choosing good notation such as using symmetrical symbols instead of asymmetrical symbols for symmetric operations, as in $x \star y$ instead of $x \to y$ (Grundy 2008, page 15);

2. some aspects are "art"-like, for example to identify the main idea of a proof and rewrite the proof to pull that idea to the foreground and push the technicalities to the background, as underlined in

   > to differentiate x^x, $\underline{\text{rewrite } x^x \text{ as } e^{x \log x}}$, then the differentiation is straightforward (by the exponential and product rules) and gives $x^x(\log x + 1)$

   instead of

$$(\underline{x^x})' = (e^{\log x^x})' = (\underline{e^{x \log x}})' = e^{x \log x}(x \log x)' = e^{\log x^x}(x \log x)' =$$
$$x^x(x \log x)' = x^x\big(1 \log x + x\tfrac{1}{x}\big) = x^x(\log x + 1).$$

1.20. Let us give a first example of a proof presentation. The example shows a proof presentation of the original proof in paragraph 1.16.

Proof.

1. We assume that G is toy secure, that is G is not constant, so there are $x, x' \in \{0,1\}^m$, necessarily distinct, such that their images $y := G(x)$ and $y' := G(x')$ are distinct.

2. Let us recall that by definition a toy breaker of $C_G = (E_G, D_G)$ is a function B such that $\forall k \in \{0,1\}^m\ \forall p \in \{0,1\}^n\ B(E_G(k,p)) = p$. So if there is a toy breaker, then each ciphertext $c = E_G(k,p)$ uniquely determines a plaintext p (independently of the key k).

3. Let us write a table with the possible

   (a) keys, emphasising x and x', in the left column;

   (b) plaintexts, emphasising y and y', in the top row;

   (c) ciphertexts, emphasising 0^n, which is the result of both encryptions

$$\begin{aligned} E_G(x\ ,y\ ) &= G(x\ ) \oplus y\ = y\ \oplus y\ = 0^n, \\ E_G(x',y') &= G(x') \oplus y' = y' \oplus y' = 0^n, \end{aligned}$$

   in the interior of the table.

The table is the following, where:

(a) "k", "p" and "c" stand respectively for "keys", "plaintexts" and "ciphertexts";

(b) ellipses stand for omitted keys, plaintexts and ciphertexts.

	p					
k	c					
		$\cdots$	y	$\cdots$	y'	$\cdots$
$\vdots$		$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
x		$\cdots$	0^n	$\cdots$	$\cdots$	$\cdots$
$\vdots$		$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
x'		$\cdots$	$\cdots$	$\cdots$	0^n	$\cdots$
$\vdots$		$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Let us notice that x and x' are in different rows because $x \neq x'$, analogously y and y' are in different columns because $y \neq y'$, so the two indicated occurrences of 0^n are in distinct cells. We can read in the table that the ciphertext 0^n results from both the distinct plaintexts y and y', so this ciphertext does not uniquely determine a plaintext, thus by point 2 there is no toy breaker, that is C_G is toy secure.

For example if $G(s) = s$, then the table is the following.

p		00	01	10	11
k	c				
00		<u>00</u>	01	10	11
01		01	<u>00</u>	11	10
10		10	11	<u>00</u>	01
11		11	10	01	<u>00</u>

We underlined in the table the ciphertexts 00. □

This proof presentation relies on two small tricks:

1. to recast the existence of a toy breaker as meaning that each ciphertext uniquely determines a plaintext;

2. to display the relation between keys, plaintexts and ciphertexts in a table.

We see a disadvantage and an advantage in this proof presentation.

1. The disadvantage is that the proof presentation is less suitable to verify for correctness (for example with the help of a computer) than the original proof because it relies on a table in which

   (a) the reader makes a visual recognition that two plaintexts y and y' encrypt to the same ciphertext 0^n;

(b) most of the table is omitted (as indicated by ellipses) and the reader has to convince himself/herself that these omissions are of no consequence for the argument being made;

(and both points are less suitable for a computer to process).

2. the advantage is that the proof presentation gives a better understanding of why the result is true than the original proof because the proof presentation

   (a) recasts the notion of a toy breaker given by the formula $\forall k \in \{0,1\}^m$ $\forall p \in \{0,1\}^n$ $B(E_G(k,p)) = p$ into the more understandable idea given essentially by the words "each ciphertext uniquely determines a plaintext (independently of the key)";

   (b) allows to "visualise the truth" of the result in a table.

1.21. Let us give a second example of a proof presentation. The example consists in rewriting the usual presentation of the classical proof that $\sqrt{2}$ is an irrational number (which does not emphasise the proof's structure and so, in principle, could be an obstacle to verifying the proof correctness) into a proof in a style closer to the one advocated by Leslie Lamport (Lamport 1995, figure 5) (emphasising structure and correctness, although our proof presentation softens his proof presentation, which employs more syntax and indentation, sometimes resembling source code). The proof presentation will have:

1. the advantage of being more suitable for a computer-assisted proof checker;

2. the disadvantage of being less suitable for a human being to read.

The original classical proof, written in the usual mathematical style, looks as follows.

Theorem. We have $\nexists r \in \mathbb{Q}\ r^2 = 2$. □

Proof. Aiming at a contradiction, let us say that there is an $r \in \mathbb{Q}$ such that $r^2 = 2$, say $r = m/n$ with $m \in \mathbb{Z}$ and $n \in \mathbb{Z} \setminus \{0\}$, where we can assume that *m and n are not both even* (by the representation of rational numbers as irreducible fractions). From $r^2 = 2$ and $r = m/n$ we get $m^2 = 2n^2$, so m is even (by Euclid's lemma $\forall p \in \mathbb{P}$ $\forall a, b \in \mathbb{Z}\ (p \mid ab \ \Rightarrow\ p \mid a \vee p \mid b)$ with $p = 2$ and $a = b = m$), that is there is a $k \in \mathbb{Z}$ such that $m = 2k$. From $m^2 = 2n^2$ and $m = 2k$ we get $n^2 = 2k^2$, so n is also even. Then m and n are both even, contradicting the assumption. □

The presentation will use mainly three devices:

1. a detailed step-by-step design presented in a linear way (although there is an underlying tree structure hidden behind the linear presentation and hinted by the division into items and subitems and by cross references);

2. the "claim method" in which

   (a) claims are raised up by stating them (as in "Claim: (a) ...");

   (b) then claims are settled down by proving them (as in "Proof: ...");

   (c) afterwards proved claims can be used at will (as in "by (a) we have...").

3. the “reference method” in which each proposition mentioned in the proof is
   (a) labelled (for example “(a)” for label (a) in the local item, and “(1a)” for label (a) in item 1);
   (b) then accessed by referencing its label.

The proof presentation is the following.

Proof.

(1) Assumptions: there is an (a) $r \in \mathbb{Q}$ such that (b) $r^2 = 2$.

Goal: (c) false.

(2) Claim: there are (a) $m \in \mathbb{Z} \wedge n \in \mathbb{Z} \setminus \{0\}$ such that (b) $r = m/n$ and (c) m and n are not both even.

Proof: by the representation of rational numbers as irreducible fractions.

(3) Claim: if (a) $e \in \mathbb{Z}$ and (b) e^2 is even, then (c) e is even.

Proof: by Euclid’s lemma $\forall p \in \mathbb{P}\ \forall a, b \in \mathbb{Z}\ (p \mid ab \ \Rightarrow\ p \mid a \vee p \mid b)$ with $p = 2$ and $a = b = e$.

(4) Claim: m is even.

Proof: we have (a) $m^2 = 2n^2$ by (1b) and (2b); we have that (b) m^2 is even by (2a) and (a); we have the claim by (2a), (b) and (3) with $e = m$.

(5) Claim: n is even.

Proof: there is a (a) $k \in \mathbb{Z}$ such that (b) $m = 2k$ by (4); we have (c) $n^2 = 2k^2$ by (1b), (2b) and (b); we have that (d) n^2 is even by (a) and (c); we have the claim by (2a), (d) and (3) with $e = n$.

(6) Claim: we have (1c).

Proof: by (2c), (4) and (5). □

1.22. Proof presentation may help solve the problem (from section 1.2) in the following three ways.

1. Proof presentation, when done in the direction of clarifying technical aspects of a proof such as
   (a) logical structure;
   (b) steps taken;
   (c) assumptions used;

   and so on, may help check the correctness of the proof.
2. Proof presentation, when done in the direction of clarifying the beautiful main ideas of a proof, may help:

(a) understanding the proof;
(b) remembering the proof;
(c) developing an intuition about the subject;
(d) inspiring other researchers.

3. Proof presentation may increase:

   (a) the audience of a proof (because it makes the proof more accessible);
   (b) the knowledge of the community (because it reaches a wider audience).

1.23. We give four examples of work on proof presentation.

1. An example of large work towards the goal of correctness is the Mizar Mathematical Library (Mizar Project 2018) of fully formalised mathematical proofs. The main idea behind the Mizar Mathematical Library is to create a huge corpus of mathematics (containing at least the fundamentals of mathematics) that:

   (a) is formalised in a language close to the language used by mathematicians;
   (b) has been checked for correctness by an interactive theorem prover;
   (c) can be extended by adding further mathematical knowledge.

2. An example of work towards both the goals of correctness and communication is Leslie Lamport's articles on how to write proofs well (Lamport 1995, Lamport 2012). Leslie Lamport advocates a hierarchical structure (making use of line breaks, indentation and labels-references as in source code) to write proofs and argues that this helps:

   (a) avoiding mistakes in proofs;
   (b) revealing the structure of proofs.

3. An example of a work mostly towards the goal of communication is Dan Grundy's PhD thesis (Grundy 2008) using as a case study a proof by Oded Goldreich (that there are weak one-way functions if and only if there are strong one-way functions). Dan Grundy:

   (a) makes an extensive analysis of the difficulty of Oded Goldreich's proof;
   (b) identifies many pitfalls (such as notation too specific or too ambiguous, unstated knowledge or assumptions, and reasoning too informal or lacking structure) common in proofs in cryptography.

4. Another example mostly towards the goal of communication is David Gries and Fred B. Schneider's book (Gries and Schneider 1993) intended to teach logic (and also discrete mathematics) by using logic in proofs in discrete mathematics instead of teaching logic in isolation. This is achieved by a proof presentation consisting in bringing logic to the foreground by means of a series of tricks whose most visible parts are changes in the usual mathematical notation and reasoning (for example by giving an equality-like treatment to equivalence through equational logic).

1.6 Solution component: automated/interactive theorem provers

1.24. An automated/interactive theorem prover is, roughly speaking, a computer program that:

1. produces a mathematical proof in the case of an automated theorem prover;
2. checks the correctness of a mathematical proof in the case of an interactive theorem prover.

In more detail:

1. an automated theorem prover
   (a) inputs
      i. the syntax of the language of a mathematical theory;
      ii. the axioms and rules of the theory;
      iii. a statement in the language;
   (b) outputs a proof of the statement (using the language and the axioms and rules) if one is found;
2. an interactive theorem prover
   (a) inputs
      i. the syntax of the language of a mathematical theory;
      ii. the axioms and rules of the theory;
      iii. a statement in the language;
      iv. a candidate to a proof of the statement using the language and the axioms and rules;
   (b) outputs an indication that the proof is correct if this is the case.

When Shai Halevi pointed out the problem (from section 1.2), he also advocated a research program to help solve the problem, consisting in creating an interactive theorem prover capable of verifying the more difficult parts of proofs and outlined the prover's capabilities and inner-workings (Halevi 2005, sections 2–5).

Automated/interactive theorem provers interact with the other solution components:

1. they can be used to prove the security of a transformation of cryptographic primitives;
2. they require rigorous definitions, theorems and proofs as does provable security;

3. but to some degree they collide with proof presentation because a proof formalised in a automated/interactive theorem prover often brings both little technical details and big ideas to the foreground ending up "not seeing the forest for the trees" (an exception being the interactive theorem prover Isabelle (University of Cambridge and Technische Universität München [Technical University of Munich] 1986), which performs better than other interactive theorem provers in terms of pushing little technical details to the background and pulling big ideas to the foreground).

1.25. An interactive/automated theorem prover may help solve the problem (from section 1.2) in the following two ways.

1. An interactive theorem prover helps:
   (a) to check the correctness of a proof if it is correct (giving the highest degree of certainty currently achievable);
   (b) to find errors in a proof if it is incorrect.
2. An automated theorem prover helps:
   (a) ideally, to automatically generate a full proof (or a counterexample, which could be an attack on a cryptographic primitive or protocol);
   (b) at least, to generate the more mechanical parts of a proof.

1.26. Let us give an example of an automated theorem prover. The example, borrowed from Ivo Seeba (Seeba 2010, sections 4 and 5.1), shows (omitting details and simplifying the syntax) how CryptoVerif proves that if $f\colon D \to D$ is an injective one-way function, then so it is its self-composition $g := f \circ f$. The one-wayness of f (apart from being polynomial-time computable) is expressed in CryptoVerif by an adversary having a negligible probability $\Pr G$ of winning the following game G (the code is on the left and the comments on the right).

G	
`new x : D`	Pick a uniformly random $x \in D$.
`adv<f(x)>`	Give $f(x)$ to the adversary.
`adv(x' : D)`	Get an $x' \in D$ from the adversary.
`if (f(x) = f(x'))`	If $f(x) = f(x')$...
`then win`	... then the adversary wins.

CryptoVerif proves the one-wayness of g by starting with a game G_1 expressing that one-wayness and rewriting G_1 into the series of games $G_1 \rightsquigarrow G_2 \rightsquigarrow G_3 \rightsquigarrow G_4$ (of which $G_1 \rightsquigarrow G_2 \rightsquigarrow G_3$ are easy transformations and $G_3 \rightsquigarrow G_4$ is the essence of the proof) below (between them there are hints for the transformation justifications)

G_1	G_2	G_3	G_4
`new x : D`	`new x : D`	`new x : D`	`new x : D`
`adv<g(x)>`	`adv<g(x)>`	`adv<f(f(x))>`	`adv<f(f(x))>`
`adv(x' : D)`	`adv(x' : D)`	`adv(x' : D)`	`adv(x' : D)`
`if (g(x) = g(x'))`	`if (x = x')`	`if (x = x')`	`if false`
`then win`	`then win`	`then win`	`then win`

g injective $\qquad g = f \circ f \qquad \Pr G \in \mathcal{N}$

such that

$$\Pr G_1 = \Pr G_2 = \Pr G_3 \approx \Pr G_4 = \underline{0}.$$

So CryptoVerif concludes $\Pr G_1 \approx \underline{0}$, that is $\Pr G_1 \in \mathcal{N}$, or in other words, g is one-way (if polynomial-time computable).

1.27. Although our work focuses on handmade proofs, we should mention that there is a developed use of automated/interactive theorem provers in cryptography, of which we give three main examples.

1. CryptoVerif is an automated theorem prover developed by Bruno Blanchet to produce security proofs of cryptographic protocols using game hopping (such as the game transformations $G_1 \rightsquigarrow G_2 \rightsquigarrow G_3 \rightsquigarrow G_4$ above) and a computational model (Blanchet 2018a), which was used to formally verify the security of a draft of TLS 1.3 (Bhargavan, Blanchet and Kobeissi 2017). In contrast to ProVerif, CryptoVerif uses a computational model (Blanchet 2016), which treats:

   (a) messages as binary strings (which are not black-boxes, so the model can look "into" messages and, for example, count the number of 0s);

   (b) cryptographic primitives as algorithms on binary strings (such as polynomial-time probabilistic algorithms).

2. ProVerif is another automated theorem prover and counter-example finder developed also by Bruno Blanchet to produce security proofs and find attacks on cryptographic protocols using a (formal-symbolic-abstract) Dolev-Yao model (Blanchet 2018b), which was also used to formally verify the security of a draft of the Transport Layer Security TLS 1.3 (Bhargavan, Blanchet and Kobeissi 2017). In contrast to CryptoVerif, ProVerif uses a Dolev-Yao model (Blanchet 2016), which treats:

   (a) cryptographic primitives (such as encryption E_A and decryption D_A for user Alice A) as black-boxes (with axioms such as $D_A(E_A(m)) = m$);

   (b) messages (between A and Bob B) as terms on primitives (such as $\big(A, D_A(E_A(m)), B\big)$ and (A, m, B), which are treated as equal by the model if and only if their equality is provable from the axioms) (Dolev and Yao 1983, section I).

3. The Tamarin Prover is an automated theorem prover and counter-example finder developed by David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse and Benedikt Schmidt to produce security proofs and find attacks on cryptographic protocols using a Dolev-Yao model (Cremers, Dreier and Sasse 2018), which was also used to formally verify the security of a draft of the TLS 1.3 (Cremers, Horvat, Hoyland, Scott and van der Merwe 2017). Like ProVerif but unlike CryptoVerif, Tamarin Prover uses a Dolev-Yao model (The Tamarin Team 2018, page 16).

CryptoVerif and ProVerif fulfil Shai Halevi's program to different extents (Barthe, Grégoire, Heraud and Béguelin 2011, page 72).

1.7 Problem format

1.28. The problems that we are interested in have the following format:

> if a first cryptographic primitive P_1 is secure according to a first security notion S_1 and it is transformed by a transformation T into a second cryptographic primitive P_2, is the second cryptographic primitive P_2 secure according to a second security notion S_2?

We can schematically represent the problem as follows:

$$P_1 \;\; S_1\text{-secure} \quad \overset{T}{\rightsquigarrow} \quad P_2 \;\; S_2\text{-secure}\,?$$

The problem has exactly the following five parameters:

$$\begin{aligned} P_1 &= \text{first cryptographic primitive,} \\ S_1 &= \text{first security notion,} \\ P_2 &= \text{second cryptographic primitive,} \\ S_2 &= \text{second security notion,} \\ T &= \text{transformation of } P_1 \text{ into } P_2. \end{aligned}$$

So an instance of the problem is completely specified by these five parameters.

1.29. Let us argue the niceness of our problem format by showing how it reformats results in cryptography into an organised structure that clarifies the role of the components of those results. For example, the classical result that there is a

> constructive proof that cryptographically-secure pseudorandom generators exist under the assumption that one-way binary-string functions exist

where we underlined the three main keywords. The last two keywords, "cryptographically-secure pseudorandom generators" and "one-way binary-string functions", are pairs of

1. security notions: "cryptographically-secure" and "one-way";
2. cryptographic primitives: "pseudorandom generators" and "binary-string functions";

so they can be further broken down as in

> constructive proof that cryptographically-secure pseudorandom generators exist under the assumption that one-way binary-string functions exist

and finally the keyword "constructive" can be recast as "transformation". This fits perfectly our problem format

$$P_1 \;\; S_1\text{-secure} \quad \overset{T}{\rightsquigarrow} \quad P_2 \;\; S_2\text{-secure}\,?$$

by letting

$$
\begin{aligned}
P_1 &= \text{binary-string function}, \\
S_1 &= \text{one-wayness}, \\
P_2 &= \text{pseudorandom generator}, \\
S_2 &= \text{cryptographic security}, \\
T &= \text{construction in the constructive proof},
\end{aligned}
$$

that is

$$
\begin{array}{rcl}
\begin{array}{r}\text{binary-string}\\ \text{function}\end{array} = P_1 & \overset{\overset{\text{construction}}{\|}}{\overset{T}{\rightsquigarrow}} & P_2 = \begin{array}{l}\text{pseudorandom}\\ \text{generator}\end{array} \\
\text{one-wayness} = S_1 & & S_2 = \begin{array}{l}\text{cryptographic}\\ \text{security}\end{array}
\end{array}
$$

1.30. We are interested in three aspects of these problems, which are the first three components of the solution from sections 1.3, 1.4 and 1.5:

1. transformation of cryptographic primitives, that is to present a suitable transformation T (of P_1 into P_2);

2. provable security, that is to prove that P_2 is S_2-secure (if P_1 is S_1-secure);

3. proof presentation, that is to give a better presentation of the security proof (mentioned in the previous point).

1.8 Overview

1.31. Let us present a chapter-by-chapter overview of this text.

Part I Introduction

Chapter 1 Introduction This chapter mainly:

1. introduces the problems that we study;
2. presents some context (background, literature and state of the art);
3. serves as a guide to the remaining of this text.

Part II Notions and notations

Chapter 2 Basics This chapter presents a collection of

1. notions;
2. notations;

that we will use.

Chapter 3 Cryptographic primitives and security notions This chapter presents a collection of formalisations of

1. cryptographic primitives;
2. security notions;

that we will use.

Part III Examples

Chapter 4 Provable security of transformation of cryptographic primitives
This chapter presents two classic examples of provable security:

1. the perfect secrecy of the one-time pad;
2. the reduction of the security of the Rabin cipher to the factorisation of Blum integers.

The examples are formatted as transformation of cryptographic primitives and security proofs of those transformations.

Chapter 5 Proof presentation of transformation of cryptographic primitives
This chapter presents two new examples of proof presentation:

1. an incorrect cryptographic proof (essentially, that if a function is collision resistant, then it is one-way) to illustrate the delicateness of this type of proofs;
2. the improvement of a proof (that if we change an arbitrary one-way function f to force $\forall n \in \mathbb{N}\ f(0^n) = \epsilon$, then f is still one-way) by using the simple notation trick of replacing a long formula $\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ by a short abbreviation $f \in \mathcal{N}$ (together with some easy and useful facts about $\mathcal{N}$).

Part IV Transformation and proof presentation

Chapter 6 Transforming one-way length-nondecreasing binary-string functions into (possibly different) one-way binary-string functions In this chapter:

1. we transform a length-nondecreasing binary-string function g into (possibly another) binary-string function $f \circ g$ by post-composition with a collision-resistant binary-string function f;
2. we prove the security of the transformation, namely that if g is one-way, then $f \circ g$ is one-way;
3. we make a proof presentation on the security proof by:
   (a) pulling the idea of the proof to the foreground;
   (b) pushing the technicalities of the proof to the background;
4. we prove the extra result that the composition of one-way binary-string functions is not necessarily one-way.

Chapter 7 Transforming cryptographically-secure pseudorandom generators into indistinguishable-from-random stream ciphers In this chapter:

1. we transform a pseudorandom generator G into a stream cipher C_G by xoring the stream of G with the plaintext given to C_G;
2. we prove the security of the transformation, namely that if G is cryptographically secure, then C_G is indistinguishable from random;

3. we make two proof presentations on the security proof:
   (a) by rewriting the proof in a schematic way;
   (b) by introducing the wedding-cake notation to simplify the presentation of formula rewritings;
4. we prove extra results:
   (a) the reciprocal implication of the security of the transformation;
   (b) indistinguishability from random implies indistinguishable encryptions;
   (c) indistinguishability from random implies semantic security;
   (d) indistinguishability from random implies bit-recovery resistance.

Chapter 8 Transforming one-way binary-string functions into NP $\setminus$ P *formal languages* In this chapter:

1. we transform a binary-string function f into a formal language $\bar{L}_f$ by roughly defining $\bar{L}_f := \{(x, f(x)) \mid x \in \{0,1\}^*\}$, or to be rigorous (since mathematical rigour is an important point of this chapter), $\bar{L}_f := \{x2f(\bar{x})21^{|\bar{x}|} \mid x \sqsubseteq \bar{x} \in \{0,1\}^*\}$;
2. we prove the security of the transformation, namely that if f is one-way, then $\bar{L}_f$ is NP $\setminus$ P;
3. we make a proof presentation on the security proof by carving out from the proof a mini-theory of minimisation operators.

Chapter 9 Transforming cryptographically-secure pseudorandom generators into one-way binary-string functions In this chapter:

1. we transform a pseudorandom generator G into a binary-string function f_G by taking $f_G(x)$ to be the stream of G of length $2|x|$ and seed x;
2. we prove the security of the transformation, namely that if G is cryptographically secure, then f_G is one-way;
3. we make a proof presentation on the security proof by converting the proof from a direct proof (a proof of $\forall x\ P(x) \Rightarrow \forall y\ Q(y)$ by assuming $\forall x\ P(x)$, taking an arbitrary y, instantiating $\forall x\ P(x)$ with an $x := t(y)$ constructed from y, and proving $Q(y)$) to a proof by reduction (a proof of $\forall x\ P(x) \Rightarrow \forall y\ Q(y)$ by taking a y such that $\neg Q(y)$, constructing an $x := t(y)$ from y, and proving $\neg P(x)$ for that particular x);
4. we prove the extra result showing that our transformation $G \rightsquigarrow f_G$ and a similar transformation $G \rightsquigarrow g_G$ by Oded Goldreich are related by one-wayness-preserving transformations $f \rightsquigarrow f_{\mathrm{p}}$ and $g \rightsquigarrow g_{\mathrm{t}}$ such that $(f_G)_{\mathrm{p}} = g_G$ and $(g_G)_{\mathrm{t}} = f_G$.

Part V Conclusion

Chapter 10 Proof presentation of transformation of cryptographic primitives This chapter presents a list of research proposals for future work. Each proposal has 3 parts:

1. the idea for the proposal;
2. some comments on the proposal;
3. the benefit of the proposal.

Chapter 11 Provable security of transformation of cryptographic primitives This chapter presents a collection of case studies on proof presentation often based on the proof presentations in the previous chapters. Each case study has six parts:

1. an introduction presenting the main idea;
2. the problem exemplified in number theory;
3. the problem exemplified in cryptography;
4. a solution exemplified in number theory;
5. a solution exemplified in cryptography;
6. a conclusion presenting the lesson learned.

1.9 Contributions

1.32. Let us indicate chapter-by-chapter our main contributions in this text.

Part I Introduction

Chapter 1 Introduction In this chapter:

1. we formulate the problems that interest us;
2. we give the toy example of the toy security of toy stream ciphers.

Part II Notions and notations

Chapter 2 Basics In this chapter we present a characterisation of negligible functions in terms of limits.

Chapter 3 Cryptographic primitives and security notions In this chapter we present:

1. less original contributions in the form of variants of know formalisations of cryptographic primitives and security notions;
2. more original contributions such as formalisations of collision resistance and stream ciphers.

Part III Examples

Chapter 4 Provable security of transformation of cryptographic primitives In this chapter we present two variants of classical security proofs:

1. a proof of the perfect secrecy of the one-time pad that is direct and does not use Bayes' theorem $\Pr[A|B] = \Pr[B|A] \ \Pr A \ / \ \Pr B$ (if $\Pr A, \Pr B \neq 0$);
2. a more formalised and yet more readable proof (than the ones that we found in the literature) of the reduction of the security of the Rabin cipher to the factorisation of Blum integers.

Chapter 5 Proof presentation of transformation of cryptographic primitives Essentially everything in this chapter is a contribution of ours (jointly in part with Eerke Boiten) but we should notice that the use of notations such as $f(n) = \text{negl}(n)$ and $f(n) \leq \varepsilon(n)$ to simplify statements and proofs, akin to our notation $f \in \mathcal{N}$, already occurs in the literature.

Part IV Transformation and proof presentation

Chapter 6 Transforming one-way length-nondecreasing binary-string functions into (possibly different) one-way binary-string functions Essentially everything in this chapter is a contribution of ours (jointly in part with Eerke Boiten) but we should notice that our counter-example to the composition of one-way binary-string functions being one-way is based on a (slightly weaker and slightly more complicated) counter-example by Pooya Farshim.

Chapter 7 Transforming cryptographically-secure pseudorandom generators into indistinguishable-from-random stream ciphers Essentially everything in this chapter is a contribution of ours except the transformation $G \rightsquigarrow C_G$, which is "folklore".

Chapter 8 Transforming one-way binary-string functions into NP \ P *formal languages* Essentially everything in this chapter is a contribution of ours but we should notice that security proof that we present:

1. was produced by us on our own and then simplified following Aaron Dutle;
2. was produced by Aaron Dutle before us but in a much less formal way (formality is a main point of this chapter).

Chapter 9 Transforming cryptographically-secure pseudorandom generators into one-way binary-string functions Essentially everything in this chapter is a contribution of ours but we should notice that Oded Goldreich has presented his transformation $G \rightsquigarrow g_G$ before we presented our transformation $G \rightsquigarrow f_G$.

Part V Conclusion

Chapter 10 Provable security of transformation of cryptographic primitives Essentially everything in this chapter is a contribution of ours in the form of (hopefully) pertinent research proposals.

Chapter 11 Proof presentation of transformation of cryptographic primitives Essentially everything in this chapter is a contribution of ours.

1.10 Conclusion

1.33. In this chapter:

1. we introduced and gave context on

   (a) transformation of cryptographic primitives;
   (b) provable security;
   (c) proof presentation;
   (d) automated/interactive theorem provers;

   and used as running examples

   (a) the toy security of toy stream ciphers;
   (b) toy one-way functions;

2. we presented an overview of the remaining text and our contributions in it.

Part II

Notions and notations

Chapter 2

Basics

2.1 Introduction

2.1. In this chapter we present a collection of notions and notations that we will need in later chapters. While mostly tedious on their own, these notions and notation do play an important role in understanding the formalisations of cryptographic primitives and security notions central to later chapters.

2.2. Let us consider, for example, the formula

$$\underbrace{\Pr T}_{(\alpha)}\Big(f_1\big(\underbrace{A(\overbrace{1^n}^{(\gamma)})}_{(\beta)}\underbrace{{}_1}_{(\delta_1)}\big),\ f_2\big(\underbrace{A(\overbrace{1^n}^{(\gamma)})}_{(\beta)}\underbrace{{}_2}_{(\delta_2)}\big)\Big) \in \underbrace{\mathcal{N}}_{(\varepsilon)},$$

where we give in addition that:

1. T is a function mapping ordered pairs of binary strings to real numbers;

2. f_1 and f_2 are functions mapping binary strings to binary strings;

3. A is a polynomial-time probabilistic algorithm that inputs binary strings and outputs ordered pairs of binary strings.

(The formula expresses that A essentially on input n outputs pairs $(x_1, x_2) := (A(1^n)_1, A(1^n)_2)$ such that the test $T(f_1(x_1), f_2(x_2))$ succeeds only with probability approaching 0 "very fast as a function of n". For example, if T tests for inequality, that is $T(y_1, y_2) \;\Leftrightarrow\; y_1 \neq y_2$, then A outputs pairs (x_1, x_2) such that $f_1(x_1) = f_2(x_2)$ holds with probability approaching 1 "very fast"; such pairs are essentially what are called *claws for f_1 and f_2*, which are of interest in cryptography.) To understand the formula, we need mainly to know the following notions and notations:

(α) $\Pr T(\ldots)$ is an abbreviation for $\Pr[T(\ldots) = 1]$, that is the probability of T outputting the "truth value" 1, or in other words, of T "succeeding";

(β) both occurrences of $A(1^n)$ denote the same output of A, not (if A is not deterministic) two (possibly different) outputs, so, for example $\Pr[A(1^n) = A(1^n)]$ means $\Pr[x = x : x \leftarrow A(1^n)]$, not $\Pr[x = y : x \leftarrow A(1^n),\, y \leftarrow A(1^n)]$;

(γ) 1^n denotes the binary string $111\cdots1$ with exactly n occurrences of the bit 1;

(δ_i) $A(1^n)_1$ and $A(1^n)_2$ denote respectively the first and second components of $A(1^n)$, so it is implicitly assumed that A outputs ordered pairs $A(x) = (A(x)_1, A(x)_2)$;

(ε) $\mathcal{N}$ denotes the set of negligible functions, which are informally functions from $\mathbb{N}$ to $\mathbb{R}$ vanishing "super-polynomially fast", and the function in question is implicitly assumed to be $\lambda n\,.\,\Pr T(\ldots)$, where n ranges over $\mathbb{N}$, so we write the shorter $\Pr T(\ldots) \in \mathcal{N}$ instead of the longer $\lambda n\,.\,\Pr T(\ldots) \in \mathcal{N}$.

These and other notions and notations are explained further below.

2.3. In this chapter, we give only the following contribution worth of notice: the equivalence

$$f \in \mathcal{N} \;\Leftrightarrow\; \forall p \lim_{n\to+\infty} p(n)f(n) = 0,$$

where

1. $f\colon L \to \mathbb{R}$ is a function with an arbitrary domain L;

2. p ranges over the positive polynomials, or equivalently, over the polynomials in one variable and with integer coefficients;

which we did not find in the literature but we saw in a talk (Comon 2016, slide 25) the somewhat similar characterisation

$$f \notin \mathcal{N} \;\Leftrightarrow\; \exists p \liminf_{n\to+\infty} p(n)f(n) > 1.$$

2.4. Although we prefer to use mathematical notation instead of English, we do sometimes write for example "$\forall A$ ..., where A ranges over the polynomial-time probabilistic algorithms outputting pairs of binary strings" instead of "$\forall A \in \mathsf{PTP}^{\text{pairs}}$ ..."; the reason for this is that we use many quantifier ranges and so it would be onerous to assign and remember notations for all of them.

2.2 Notions and notations

2.5. In the following definition we collect various notions and notations.

2.6 Definition.

Sets of numbers

1. $\mathbb{N}$ denotes the *set of natural numbers* $\{0, 1, 2, 3, 4, \ldots\}$.
2. $\mathbb{N} \setminus \{0\}$ denotes the *set of positive natural numbers* $\{1, 2, 3, 4, \ldots\}$.
3. $\mathbb{P}$ denotes the *set of prime numbers* $\{2, 3, 5, 7, 11, \ldots\}$.
4. $[m\mathinner{..}n]$ denotes the *integer interval* $\{i \in \mathbb{N} \mid m \leq i \leq n\}$ from m (inclusive) to n (inclusive) (where $m, n \in \mathbb{N}$).

Relations on numbers

1. $|$ (as in $m \mid n$) denotes the *divisibility relation*.

Functions on numbers

1. $\lfloor \cdot \rfloor$ denotes the *floor function* $\lfloor x \rfloor := \max\{i \in \mathbb{Z} \mid i \leq x\}$ (where $x \in \mathbb{R}$).
2. A *positive polynomial* (Delfs and Knebl 2007, definition 5.4) is a polynomial $p(n)$ in one variable n and with coefficients in $\mathbb{Z}$ such that $\forall n \in \mathbb{N} \setminus \{0\}$ $p(n) > 0$.

Logic

1. $\operatorname{Tr} P$ denotes the *truth value* (0 or 1) of P (where P is a predicate).

Sets

1. $\mathcal{P}(X)$ denotes the *powerset* $\{Y \mid Y \subseteq X\}$ of X (where X is a set).

Functions on functions

1. $f^{-1}[y]$ denotes the *set of preimages* $\{x \in X \mid f(x) = y\}$ of y under f (where $f\colon X \to Y$ is a function and y is arbitrary).

   Let us notice that $f^{-1}[y]$ is a set of elements and not a specific element such as a *preimage* of y under f (that is an $x \in X$ such that $f(x) = y$) nor (if f is bijective and $y \in Y$) the *inverse* $f^{-1}(y)$ of y under f (that is the unique $x \in X$ such that $f(x) = y$).
2. $\operatorname{dom}(f)$ denotes the *domain* of f (where f is a function).
3. $\operatorname{im}(f)$ denotes the *image* of f (where f is a function).

Relations on functions

1. $f \leq g$ means $\forall n \in \mathbb{N}\ f(n) \leq g(n)$ (where $f, g\colon \mathbb{N} \to \mathbb{R}$ are functions).

Negligibility

1. A *negligible function* (Goldreich 2004, definition 1.3.5) is a function $f\colon L \to \mathbb{R}$ with an arbitrary domain L (but only of interest when $\mathbb{N} \cap L$ is infinite) such that

$$\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$$

   (where p ranges over the positive polynomials, N ranges over $\mathbb{N}$, and n ranges over $\mathbb{N} \cap L$), or equivalently, such that

$$\forall p \lim_{n \to +\infty} p(n) f(n) = 0$$

(where p ranges over the positive polynomials, or equivalently, over the polynomials in one variable and with integer coefficients, and the limit means

$$\forall \varepsilon > 0 \; \exists N \in \mathbb{N} \; \forall n \in \mathbb{N} \cap L \; (n > N \;\Rightarrow\; |p(n)f(n) - 0| < \varepsilon),$$

where we replaced the usual $n \in \mathbb{N}$ by $n \in \mathbb{N} \cap L$ for $f(n)$ to be defined).

We often say, for example, that 2^{-n} is negligible to mean that the function $f\colon \mathbb{N} \to \mathbb{R}$ defined by $f(n) := 2^{-n}$, that is $\lambda n \,.\, 2^{-n}$ where n ranges in $\mathbb{N}$, is negligible.

2. $\mathcal{N}$ denotes the *set of negligible functions.*

   We often write, for example, $2^{-n} \in \mathcal{N}$ to mean $f \in \mathcal{N}$ where $f\colon \mathbb{N} \to \mathbb{R}$ is the function defined by $f(n) := 2^{-n}$, that is $\lambda n \,.\, 2^{-n}$ where n ranges in $\mathbb{N}$.

   Sometimes we abuse the notation and write for example $P(n) := 2^{-n^2} \leq 2^{-n} \in \mathcal{N}$ to mean $P(n) := 2^{-n^2} \wedge \forall n \in \mathbb{N} \; 2^{-n^2} \leq 2^{-n} \wedge \lambda n \,.\, 2^{-n} \in \mathcal{N}$.

   We have $f \in \mathcal{N} \;\Leftrightarrow\; |f| \in \mathcal{N}$, so we write $f \in \mathcal{N}$ in situations where one may expect $|f| \in \mathcal{N}$.

3. $f\colon A \nrightarrow B$ denotes a *partial function* f from A to B (where A and B are sets).

4. f is *two-to-one* if and only if every image of f has exactly two distinct preimages, that is $\forall y \in \mathrm{im}(f) \; |f^{-1}[y]| = 2$ (where f is a function).

Sets of strings

1. $\{0,1\}^n$ denotes the *set of binary strings of length* n (where $n \in \mathbb{N}$).

   Analogously for X^n (where X is a set).

2. $\{0,1\}^{\leq n}$ denotes the *set of binary strings of length less than or equal to* n (where $n \in \mathbb{N}$).

   Analogously for $X^{\leq n}$ (where X is a set).

3. $\{0,1\}^*$ denotes the *set of binary strings (of finite length).*

   Analogously for X^* (where X is a set).

4. $\{0,1\}^{2*} := \bigcup_{n \in \mathbb{N}} \{0,1\}^{2n}$ denotes the *set of binary strings of even length.*

   Analogously for X^{2*} (where X is a set).

Special strings

1. ϵ denotes the *empty string.*

2. 0^n denotes the *all-zero binary-string of length* n, that is $\underbrace{000\ldots0}_{n \text{ zeros}}$ (where $n \in \mathbb{N}$).

3. 1^n denotes the *all-one binary-string of length* n, that is $\underbrace{111\ldots1}_{n \text{ ones}}$ (where $n \in \mathbb{N}$).

Functions on strings

1. $|x|$ denotes the *length* of x (where $x \in \{0,1\}^*$).
2. $|(x_1, \dots, x_n)|$ denotes $|x_1 \dots x_n| = |x_1| + \dots + |x_n|$ (where $n \in \mathbb{N}$ and $x_1, \dots, x_n \in \{0,1\}^*$).
3. If $x = x_1 \dots x_n$ with $x_1, \dots, x_n \in \{0,1\}$ and $n \in \mathbb{N}$, then

$$x^i := \begin{cases} 0 & \text{if } n = 0 \\ \begin{cases} x_1 & \text{if } i = 0 \\ x_i & \text{if } 1 \leq i \leq n \\ x_n & \text{if } i > n \end{cases} & \text{if } n \neq 0 \end{cases}$$

   (where $i \in \mathbb{N}$).
4. xy denotes the *concatenation* of x and y (where $x, y \in \{0,1\}^*$).
5. $x|_n$ denotes the *restriction* of x to its prefix of length n (if $n > |x|$, then $x|_n = x$) (where $x \in \{0,1\}^*$ and $n \in \mathbb{N}$).
6. $x_{\leftharpoondown}$ denotes the *left half* $x_1 \dots x_{\lfloor n/2 \rfloor}$ of $x = x_1 \dots x_n$ (where $n \in \mathbb{N}$ and $x_1, \dots, x_n \in \{0,1\}$).
7. $x_{\rightharpoondown}$ denotes the *right half* $x_{\lfloor n/2 \rfloor + 1} \dots x_n$ of $x = x_1 \dots x_n$ (where $n \in \mathbb{N}$ and $x_1, \dots, x_n \in \{0,1\}$).
8. $x_{\twoheadleftarrow}$ denotes the *init* (or *head*)

$$\begin{cases} x_1 \dots x_{n-1} & \text{if } n > 1 \\ \epsilon & \text{if } n \leq 1 \end{cases}$$

   of $x = x_1 \dots x_n$ (where $n \in \mathbb{N}$ and $x_1, \dots, x_n \in \{0,1\}$).
9. $x_{\twoheadrightarrow}$ denotes the *tail*

$$\begin{cases} x_n & \text{if } n \geq 1 \\ \epsilon & \text{if } n = 0 \end{cases}$$

   of $x = x_1 \dots x_n$ (where $n \in \mathbb{N}$ and $x_1, \dots, x_n \in \{0,1\}$).
10. n_2 denotes the binary string obtained by writing n in *base 2* without leading zeros (where $n \in \mathbb{N}$).
11. $(n_1, \dots, n_k)_2$ denotes $((n_1)_2, \dots, (n_k)_2)$ (where $k, n_1, \dots, n_k \in \mathbb{N}$). Analogously for sets instead of tuples.
12. ${}_2x$ denotes:
    (a) the natural number obtained by interpreting x as a natural number written in *base 2* if $x \in \{0,1\}^* \setminus \{\epsilon\}$;
    (b) the natural number 0 if $x = \epsilon$.
13. ${}_2(x_1, \dots, x_k)$ denotes $({}_2(x_1), \dots, {}_2(x_k))$ (where $k \in \mathbb{N}$ and $x_1, \dots, x_k \in \{0,1\}^*$). Analogously for sets instead of tuples.

14. $\oplus$ denotes *xor* defined by the following equalities or table

$$\oplus = +_2 = -_2, \qquad \begin{aligned} 0 \oplus 0 &= 0, \\ 0 \oplus 1 &= 1, \\ 1 \oplus 0 &= 1, \\ 1 \oplus 1 &= 0, \end{aligned} \qquad \begin{array}{c|cc} \oplus & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}$$

(where $+_2$ and $-_2$ denote respectively addition modulo 2).

15. $\oplus$ also denotes the *bitwise* (*bit-by-bit*) xor of two binary strings of the same length, that is

$$(x_1 \dots x_n) \oplus (y_1 \dots y_n) = (x_1 \oplus y_1) \dots (x_n \oplus y_n)$$

(where $n \in \mathbb{N}$ and $x_1 \dots x_n, y_1 \dots y_n \in \{0, 1\}$).

16. $\ominus$ denotes $\oplus$ (which is explained by the fact that we can define $\oplus := +_2$ and $\ominus := -_2$, and that it happens here in particular that $+_2 = -_2$ and more generally in all rings of characteristic 2) (where $+_2$ and $-_2$ denote respectively addition modulo 2).

17. $x[y \leftarrow z]$ denotes the result of *simultaneously substituting* (*replacing*) in x all occurrences of z by y (where $x, y, z \in \{0, 1\}^*$).

18. A (total) *binary-string function* is a function $f\colon \{0, 1\}^* \to \{0, 1\}^*$.

19. A *partial binary-string function* is a function $f\colon \bigcup_{l \in L} \{0, 1\}^l \to \{0, 1\}^*$ (where $L \subseteq \mathbb{N}$ is infinite).

20. f is *length preserving* (Goldreich 2004, definition 2.2.4) if and only if $\forall x \in \mathrm{dom}(f)\ |x| = |f(x)|$ (where f is a partial binary-string function).

21. f is *length nondecreasing* if and only if $\forall x \in \mathrm{dom}(f)\ |x| \leq |f(x)|$ (where f is a partial binary-string function).

22. f is *length regular* (Goldreich 2004, section 2.2.3.2) if and only if

$$\forall x, x', y, y' \in \{0, 1\}^* \ (|y| = |y'| \ \Rightarrow \ |f(x, y)| = |f(x', y')|)$$

(where $f\colon \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^*$ is a function).

23. f is *polynomial-to-one* (Boiten and Grundy 2010, q-pre-injectiveness in paragraph "One-way functions and pre-composition") if and only if every image of f has at most polynomially many preimages, that is there is a positive polynomial p such that $\forall x \in \{0, 1\}^*\ |f^{-1}[x]| \leq p(|x|)$ (where f is a binary-string function).

24. f *does not depend on the left half of the input* if and only if

$$\forall x, x' \in \mathrm{dom}(f) \ (x_{\leftarrow} = x'_{\leftarrow} \ \Rightarrow \ f(x) = f(x'))$$

(where f is a partial binary-string function).

25. f *does not depend on the right half of the input* if and only if

$$\forall x, x' \in \mathrm{dom}(f) \ (x_{\rightarrow} = x'_{\rightarrow} \ \Rightarrow \ f(x) = f(x'))$$

(where f is a partial binary-string function).

Relations on strings

1. $\sqsubseteq$ (as in $x \sqsubseteq y$) denotes the *prefix* (*initial segment*) relation (where $x, y \in \{0,1\}^*$).
2. $\leq_{\text{lex}}$ denotes the *non-strict lexicographic order in* $\{0,1\}^*$ (Avigad and Goldreich 2011, section 4.1) defined by the following condition: if

$$\begin{aligned} x &= x_1 \dots x_m, \quad x_1, \dots, x_m \in \{0,1\}, \quad m \in \mathbb{N}, \\ y &= y_1 \dots y_n, \qquad y_1, \dots, y_n \in \{0,1\}, \quad n \in \mathbb{N}, \end{aligned}$$

$$\begin{aligned} i_{\min}{\downarrow} &:\Leftrightarrow \exists i \in [1 \mathinner{..} \min(m,n)]\ x_i \neq y_i, \\ i_{\min} &:= \min\{i \in [1 \mathinner{..} \min(m,n)] \mid x_i \neq y_i\} \end{aligned}$$

(so $i_{\min}$ is defined if and only if $i_{\min}{\downarrow}$), then

$$x \leq_{\text{lex}} y \ :\Leftrightarrow\ x \sqsubseteq y \lor (i_{\min}{\downarrow} \land x_{i_{\min}} < y_{i_{\min}}).$$

3. $\min_{\text{lex}} X$ denotes the *minimum* (if it exists) of X with respect to $\leq_{\text{lex}}$ (where $X \subseteq \{0,1\}^*$).

Probability

1. U_n and U'_n denote *uniform random variables* in $\{0,1\}^n$ (where $n \in \mathbb{N}$).
2. If in a formula there are two occurrences of a random variable X, for example $\Pr[f(X) = f(X)]$, then they denote the same outcome of X (Goldreich 2004, pages 8–9 and 34).

   For example $\Pr[f(X) = f(X)]$ means $\Pr[f(x) = f(x) : x \leftarrow X]$ and not $\Pr[f(x) = f(x') : x \leftarrow X, x' \leftarrow X]$ (the latter would be denoted by $\Pr[f(X) = f(X')]$ with X' being a random variable identically distributed to X but independent from X).

   Analogously, if X is a probabilistic algorithm instead of a random variable.
3. $\Pr A(\vec{x})$ is an abbreviation of $\Pr[A(\vec{x})]$, which in turn is an abbreviation of $\Pr[A(\vec{x}) = 1]$ (where A is an algorithm and $\vec{x}$ is a tuple of binary strings).
4. $\Pr E$ is an abbreviation of $\Pr[E]$ (where E is an event).

Algorithms

1. P is the *class* (set) *of formal languages decidable by polynomial-time deterministic algorithms* (Turing machines) (Papadimitriou 1994, section 2.4).
2. NP is the *class* (set) *of formal languages decidable by polynomial-time nondeterministic algorithms* (Turing machines) (Papadimitriou 1994, section 2.7).
3. $A(1^n) = U_n$ means that the polynomial-time probabilistic algorithm A is such that $|A(1^n)| = n$ and $\forall x \in \{0,1\}^n\ \Pr[A(1^n) = x] = 2^{-n}$, that is $A(1^n)$ takes values in $\{0,1\}^n$ and (considered as a random variable) is uniformly distributed as U_n (where $n \in \mathbb{N}$).

4. If A on input $\vec{x}$ outputs ordered pairs of binary strings, then $A(\vec{x})_1$ and $A(\vec{x})_2$ denote respectively the first and second components of the ordered pairs, that is $A(\vec{x}) = (A(\vec{x})_1, A(\vec{x})_2)$ (where A is an algorithm and $\vec{x}$ is a tuple of binary strings). We often simply write $A(\vec{x})_1$ or $A(\vec{x})_2$ without explicitly saying that A outputs ordered pairs, and in such cases it is implicitly understood that A does output ordered pairs (we mean exactly 2-tuples, not 3-tuples nor 4-tuples nor so on) of binary strings. Analogously, if A outputs n-tuples (where $n \in \mathbb{N} \setminus \{0\}$).
5. $y{\downarrow}$ and $y = {\downarrow}$ mean that the object y is *defined*. For example $f(x){\downarrow}$ and $f(x) = {\downarrow}$ mean that x is in the domain of the partial function f, and $A(x){\downarrow}$ and $A(x) = {\downarrow}$ mean that x is an input for which the algorithm A halts.
6. $y{\uparrow}$ and $y = {\uparrow}$ mean that the object y is *undefined*. For example $f(x){\uparrow}$ and $f(x) = {\uparrow}$ mean that x is not in the domain of the partial function f, and $A(x){\uparrow}$ and $A(x) = {\uparrow}$ mean that x is an input for which the algorithm A does not halt.
7. $x = y{\downarrow}$ means that x and y are both defined and are equal, that is $x{\downarrow} \wedge y{\downarrow} \wedge x = y$.
8. $x \overset{\downarrow\uparrow}{=} y$ means that x and y are both defined and are equal, or they are both undefined, that is $(x{\downarrow} \wedge y{\downarrow} \wedge x = y) \vee (x{\uparrow} \wedge y{\uparrow})$.

2.3 Conclusion

2.7. In this chapter we presented a collection of notions and notations that we will need in later chapters, of which perhaps the more important ones are:

1. $\Pr A(x)$ abbreviates $\Pr[A(x) = 1]$;
2. multiple occurrences of $A(x)$ and U_n denote the same outcome;
3. 1^n denotes $111\ldots1$ with exactly n times the bit 1;
4. $A(x)_i$ denotes the i-th component of $A(x)$;
5. the set $\mathcal{N}$ of negligible functions.

Chapter 3

Cryptographic primitives and security notions

3.1 Introduction

3.1. This chapter presents a collection of formalisations of cryptographic primitives and security notions that we will need in later chapters. (They are all collected here, instead of appearing in the chapters where they are needed, to avoid duplication because some on them are needed in more than one chapter.)

3.2. In this chapter, we give the following contributions.

Less original contributions Our own formalisations that we found already partially or fully formalised in a similar way in the literature:

1. perfect secrecy;
2. cryptographic security;
3. indistinguishable encryption;
4. semantic security;
5. bit-recovery resistance.

More original contributions Our own formalisations that we did not find already partially nor fully formalised in a similar way in the literature:

1. collision resistance (jointly with Eerke Boiten);
2. stream cipher.

3.3. This chapter includes work taken from the following two articles:

1. the notion of collision resistance and sections 3.11 and 3.12 below are based on joint work with Eerke Boiten (Gaspar and Boiten 2014, section 1.3).
2. the notion of indistinguishability from random below is based on a informally published work of ours (Gaspar 2016, definition 5).

3.2 Primitive: random generator

3.4. An n-length random generator is informally an algorithm that inputs n and outputs a random string of length n.

3.5 Definition. Let $n \in \mathbb{N}$. An *n-length random generator* (Goldreich 2004, adapted from definition 3.3.4) is a polynomial-time probabilistic algorithm G_n such that $\Pr[|G_n(1^n)| = n] = 1$, that is $|G_n(1^n)| = n$ for all runs of G_n.

3.6. We could have defined a random generator as being a polynomial-time probabilistic algorithm G such that $\forall n \in \mathbb{N}\ |G(1^n)| = n$ and then obtain from G the n-length random generator $G_n(x) := G(1^n)$ (where $x \in \{0,1\}^*$), but in the notion of n-length one-time pad it is convenient to have a parameter n, so we do the same for G_n.

We could also have "hardwired" the input 1^n in G_n getting $G' := G_n(1^n)$, but we have to make the input explicit so that we can require G_n to run in polynomial time in the length of the input.

3.3 Security: uniformity

3.7. The n-length uniformity of an n-length random generator G_n informally means that the outputs of G_n are unbiased.

3.8 Definition. Let $n \in \mathbb{N}$. An n-length random generator G_n is *n-length uniform* (Goldreich 2004, section 1.2.1) if and only if $G(1^n) = U_n$, that is the probability distribution of $G(1^n)$ is uniform.

3.4 Primitive: one-time pad

3.9. The n-length one-time pad C_{n,G_n} induced by the n-length random generator G_n is informally the cipher that:

1. generates a key $k := G_n(1^n)$ (independently of the plaintexts);
2. encrypts a plaintext p to $k \oplus p$;
3. decrypts a ciphertext c to $k \oplus c$.

3.10 Definition. Let $n \in \mathbb{N}$.

1. The *n-length one-time pad C_G induced by the n-length random generator G_n* (Buchmann 2001, section 4.5) is the 6-tuple $C_{n,G_n} := (\mathcal{K}_n, \mathcal{P}_n, \mathcal{C}_n, K_{n,G_n}, E_n, D_n)$ where:
   (a) $\mathcal{K}_n := \{0,1\}^n$ is called *key space*;
   (b) $\mathcal{P}_n := \{0,1\}^n$ is called *plaintext space*;
   (c) $\mathcal{C}_n := \{0,1\}^n$ is called *ciphertext space*;
   (d) $K_{n,G_n} := G_n$ is called *key generator*;

(e) E_n defined by

$$\begin{aligned} E_n \colon \mathcal{K}_n \times \mathcal{P}_n &\to \mathcal{C}_n \\ (k, p) &\mapsto k \oplus p \end{aligned}$$

is called *encryption function*;

(f) D_n defined by

$$\begin{aligned} D_n \colon \mathcal{K}_n \times \mathcal{C}_n &\to \mathcal{P}_n \\ (k, c) &\mapsto k \oplus c \end{aligned}$$

is called *decryption function*.

2. We:

   (a) define a probability distribution $\Pr_{\mathcal{K}_n} k := \Pr[K_{n,G_n}(1^n) = k]$ on $\mathcal{K}_n$;

   (b) assume that there is an arbitrary probability distribution $\Pr_{\mathcal{P}_n}$ on $\mathcal{P}_n$;

   (c) assume that $\Pr_{\mathcal{K}_n}$ and $\Pr_{\mathcal{P}_n}$ are independent and so induce a probability distribution $\Pr_{\mathcal{K}_n \times \mathcal{P}_n} (k, p) := \Pr_{\mathcal{K}_n} k \times \Pr_{\mathcal{P}_n} p$ in $\mathcal{K}_n \times \mathcal{P}_n$.

3.5 Security: perfect secrecy

3.11. Before we progress, we need to make a small digression into probability theory to further discuss $\Pr_{\mathcal{K}_n \times \mathcal{P}_n}$ so that we can talk simultaneously of the probability of events in $\mathcal{K}_n$ and events in $\mathcal{P}_n$. Let $n \in \mathbb{N}$.

1. We consider the variables:

   (a) k ranging over $\mathcal{K}_n$;

   (b) p ranging over $\mathcal{P}_n$.

2. We define:

   (a) $P_p := \mathcal{K}_n \times \{p\}$, which is the event "the plaintext is p" in $\mathcal{K}_n \times \mathcal{P}_n$;

   (b) $C_c := \{(k', p') \in \mathcal{K}_n \times \mathcal{P}_n \mid E_n(k', p') = c\}$, which is the event "the ciphertext is c" in $\mathcal{K}_n \times \mathcal{P}_n$.

   (There is no need to add n as a subscript to P_p and C_c because n can be determined from p and c since $|p| = n = |c|$.)

3. We adopt the following abbreviations:

   (a) $\Pr p := \Pr_{\mathcal{K}_n \times \mathcal{P}_n} P_p$, which is the probability of the event "the plaintext is p" in $\mathcal{K}_n \times \mathcal{P}_n$ and is equal to $\Pr_{\mathcal{P}_n} p$;

   (b) $\Pr[p|c] := \Pr_{\mathcal{K}_n \times \mathcal{P}_n} P_p \cap C_c \,/\, \Pr_{\mathcal{K}_n \times \mathcal{P}_n} C_c$, which is the conditional probability of the event "the plaintext is p" in $\mathcal{K}_n \times \mathcal{P}_n$ given the event "the ciphertext is c" in $\mathcal{K}_n \times \mathcal{P}_n$.

   (We have $\Pr_{\mathcal{K}_n \times \mathcal{P}_n} C_c > 0$, as required for the conditional probability to be defined, if $\Pr_{\mathcal{K}_n}$ is uniform: taking a $p' \in \mathcal{P}_n$ such that $\Pr_{\mathcal{P}_n} p' > 0$, and taking $k' := p' \oplus c$, which is such that $\Pr_{\mathcal{K}_n} k' = 1/|\mathcal{K}_n| > 0$, we have $(k', p') \in C_c$, so $\Pr_{\mathcal{K}_n \times \mathcal{P}_n} C_c \geq \Pr_{\mathcal{K}_n \times \mathcal{P}_n} (k', p') = \Pr_{\mathcal{K}_n} k' \times \Pr_{\mathcal{P}_n} p' > 0$.)

3.12. The n-length perfect secrecy of an n-length one-time pad C_{n,G_n} informally means that ciphertexts c computed by C_{n,G_n} do not reveal any information about their corresponding plaintexts p, in the sense that learning c does not improve the probability of discovering p, that is $\Pr p = \Pr[p|c]$.

3.13 Definition. Let $n \in \mathbb{N}$. An n-length one-time pad C_{n,G_n} is *n-length perfectly secret* (Buchmann 2001, definition 4.4.1) if and only if

$$\forall p, c \;\; \Pr p = \Pr[p|c],$$

where

1. p ranges over $\mathcal{P}_n$;

2. c ranges over $\mathcal{C}_n$.

3.6 Primitive: Blum-integer factorisation problem

3.14. Before we progress, we need to make a small digression into number theory discussing Blum primes and Blum integers.

1. A *Blum prime* (Talbot and Welsh 2006, problem 6.10) is a $q \in \mathbb{P}$ such that $q \equiv 3 \pmod 4$.

   (Blum primes are named after Manuel Blum, possibly because they are used in his Blum-Blum-Shub pseudorandom generator and in his Blum-Goldwasser cipher. At the "limit of infinity", "half" of the prime numbers are Blum primes, or more rigorously, if $\pi_i(n) := |\{q \in \mathbb{P} \mid n \geq q \equiv i \pmod 4\}|$, then $\lim_{n\to+\infty} \pi_1(n)/\pi_3(n) = 1$ (Granville and Martin 2006, formula (1)). But it is conjectured that before the "limit of infinity", primes numbers are biased to be Blum primes, or more rigorous but not completely rigorous, $\pi_1(n) < \pi_3(n)$ occurs more often than $\pi_1(n) > \pi_3(n)$ (Granville and Martin 2006, first quotation on page 2), something known as Khrushchev's bias (Granville and Martin 2006, last quotation on page 22).)

2. A *Blum integer* (Menezes, van Oorschot and Vanstone 1996, definition 2.156) is the product of two distinct Blum primes q and q'.

   (Blum integers qq' are of particular interest in number theory and its applications to cryptography because they make it easier to calculate square roots modulo qq'.)

3.15. The Blum-integer factorisation problem is informally the problem of presenting an algorithm A that factors Blum integers (at least half of the time), that is A inputs (in base 2) a Blum integer qq' and outputs (in base 2) q or q' (with probability at least 1/2).

3.16 Definition. The *Blum-integer factorisation problem* is the following problem: to present a probabilistic algorithm A such that

$$\forall qq' \ \Pr[A((qq')_2) \in \{q_2, q'_2\}] \geq 1/2,$$

where qq' ranges over the Blum integers (and q and q' are the Blum prime factors of qq').

3.7 Security: Blum-integer factorisation problem hardness

3.17. The hardness of the Blum-integer factorisation problem informally means that the problem has no easy solution, when by an easy solution we understand a polynomial-time probabilistic algorithm.

3.18 Definition. We say that the Blum-integer factorisation problem is *hard* if and only if no polynomial-time probabilistic algorithm A solves the Blum-integer factorisation problem.

3.8 Primitive: Rabin cipher

3.19. Before we progress, we need to make a small digression into number theory discussing quadratic residues modulo n and modular square roots modulo n. Let $n \in \mathbb{N}$.

1. A *quadratic residue modulo* n (Menezes, van Oorschot and Vanstone 1996, page 209) is a $c \in \mathbb{N}$ such that there is a $p \in \mathbb{N}$ such that $c \equiv p^2 \pmod{n}$.

2. A *square root of* $c \in \mathbb{N}$ *modulo* n is a $p \in \mathbb{N}$ such that $c \equiv p^2 \pmod{n}$.

3. Let $c \in \mathbb{N}$ and let us now restrict ourselves to the case where $n = qq'$ is a Blum integer, so $c^{(q+1)/4}, c^{(q'+1)/4} \in \mathbb{N}$ because $(q+1)/4, (q'+1)/4 \in \mathbb{N}$. We define

   $$\sqrt{c} \bmod qq'_{\pm\mp} := (\pm c^{(q'+1)/4} q^{q'-1} \mp c^{(q+1)/4} q'^{q-1}) \bmod qq',$$

   where the signs $\pm$ and $\mp$ are independent, so there are 4 combinations of signs, and we denote the set/tuple of the 4 values (in some order) $\sqrt{c} \bmod qq'_{\pm\mp}$ by $\sqrt{c} \bmod qq'$. Then the elements of $\sqrt{c} \bmod qq'$ are the square roots of c modulo qq', or more precisely, we make the claim (Buchmann 2001, based on section 7.3.3)

   > the square roots of a quadratic residue $c \equiv p^2 \pmod{qq'}$ modulo qq' are modulo qq' exactly the elements of $\sqrt{c} \bmod qq'$

   proved in proof 3.24. (If q and q' are large, then $\sqrt{c} \bmod qq'_{\pm\mp}$ may be very large, even not polynomial-time computable in $|(qq')_2|$, so we included the part "… mod qq'" in its definition; in calculations we may want to use instead the variant

$$\sqrt{c}' \bmod qq'_{\pm\mp} := ((\pm c^{(q'+1)/4} q^{q'-2} \bmod q')q + (\mp c^{(q+1)/4} q'^{q-2} \bmod q)q') \bmod qq'$$

and even make more reductions such as substituting $c^{(q'+1)/4} q^{q'-2} \bmod q'$ by $(c^{(q'+1)/4} \bmod q')(q^{q'-2} \bmod q')$; proof 3.24 still holds for the variant.)

3.20. The Rabin cipher is informally the cipher that:

1. generates as key a Blum integer qq';
2. encrypts a plaintext p by squaring it to $p^2 \bmod qq'$;
3. decrypts a ciphertext c by square rooting it to $\sqrt{c} \bmod qq'$.

3.21 Definition. The *Rabin cipher* C (Buchmann 2001, based on sections 7.3.1–7.3.3) is the 7-tuple $C := (\mathcal{K}, \mathcal{K}', \mathcal{P}, \mathcal{C}, K, E, D)$ where

1. $\mathcal{K} := \{(q, q') \in \mathbb{P} \times \mathbb{P} \mid q \neq q' \ \wedge \ q, q' \equiv 3 \pmod 4\}$ is called *private-key space*;
2. $\mathcal{K}' := \{qq' \mid (q, q') \in \mathcal{K}\}$ is called *public-key space*;
3. $\mathcal{P} := \mathbb{N}$ is called *plaintext space*;
4. $\mathcal{C} := \mathbb{N}$ is called *ciphertext space*;
5. K is a probabilistic algorithm, such that on input $n \in \mathbb{N}$ outputs $((q, q'), qq') \in \mathcal{K} \times \mathcal{K}'$ with $qq' \geq n$, called *key generator*;
6. E defined by
$$\begin{aligned} E\colon \bigcup_{qq' \in \mathcal{K}'} (\{qq'\} \times [0 \mathinner{..} qq' - 1]) &\to \mathcal{C} \\ (qq', p) &\mapsto p^2 \bmod qq' \end{aligned}$$
is called *encryption function*;
7. D defined by
$$\begin{aligned} D\colon \mathcal{K} \times \mathbb{N} &\to \mathbb{N}^4 \\ ((q, q'), c) &\mapsto \sqrt{c} \bmod qq' \end{aligned}$$
is called *decryption function*.

3.22. The decryption D of the Rabin cipher does not produce the plaintext p, that is $p \neq D((q, q'), E(qq', p))$, but a set/tuple of 4 possible plaintexts with the guarantee that one of them is the correct plaintext, that is $p \in D((q, q'), E(qq', p))$, so there is ambiguity in decryption. One way to resolve the ambiguity would be to order the 4 possible plaintexts (they are natural numbers, so they can be ordered canonically) and to send along with the ciphertext the index of the correct plaintext (this would not break the security proof of the Rabin cipher).

3.23. As promised, we give a proof of the claim made in paragraph 3.19.

3.24 Proof.

1. First, we make two claims.

   (a) Claim: p is a square root of c modulo qq' if and only if p is a square root of c modulo q and q'.

   Proof: $c \equiv p^2 \pmod{qq'} \Leftrightarrow qq' \mid c - p^2 \Leftrightarrow q \mid c - p^2 \wedge q' \mid c - p^2 \Leftrightarrow c \equiv p^2 \pmod{q} \wedge c \equiv p^2 \pmod{q'}$ using $q, q' \in \mathbb{P}$ and $q \neq q'$.

   (b) Claim: two square roots p and p' of c modulo q differ at most in their signs modulo q; analogously for q' instead of q.

   Proof: $c \equiv p^2 \pmod{q} \wedge c \equiv p'^2 \pmod{q} \Rightarrow p^2 \equiv p'^2 \pmod{q} \Leftrightarrow q \mid p^2 - p'^2 = (p - p')(p + p') \Leftrightarrow q \mid p - p' \vee q \mid p + p' \Leftrightarrow p \equiv +p' \pmod{q} \vee p \equiv -p' \pmod{q}$ by Euclid's lemma $\forall q \in \mathbb{P}\ \forall a, b \in \mathbb{Z}\ (q \mid ab \Rightarrow q \mid a \vee q \mid b)$.

2. Second, in the next two claims we check that the square roots of c modulo qq' are exactly the elements of $\sqrt{c} \bmod qq'$ modulo qq' using the previous claims.

   (a) Claim: all elements of $\sqrt{c} \bmod qq'$ are square roots of c modulo q, that is for each element e, we have $c \equiv e^2 \pmod{q}$; analogously for modulo q' instead of q.

   Proof: $(\sqrt{c} \bmod qq'_{++})^2 \equiv (c^{(q+1)/4} q'^{q-1})^2 = c^{(q+1)/2} (q'^{q-1})^2 \equiv (p^2)^{(q+1)/2} (q'^{q-1})^2 = pp^q (q'^{q-1})^2 \equiv p \cdot p \cdot 1^2 = p^2 \equiv c \pmod{q}$ by $\forall a \in \mathbb{Z}\ \forall n \in \mathbb{N} \setminus \{0\}\ a \bmod n \equiv a \pmod{n}$, $q \mid q^{q'-1}$ and Fermat's little theorem $\forall q \in \mathbb{P}\ \forall a \in \mathbb{Z}\ \big(a^q \equiv a \pmod{q} \wedge (q \nmid a \Rightarrow a^{q-1} \equiv 1 \pmod{q})\big)$; analogously for the other elements.

   (b) Claim: the elements of $\sqrt{c} \bmod qq'$ contain all possible signs modulo q, that is for each element e there is an element e' such that $e \equiv -e' \pmod{q}$; analogously for q' instead of q.

   Proof: $\sqrt{c} \bmod qq'_{++} \equiv +c^{(q+1)/4} q'^{q-1} = -(-c^{(q+1)/4} q'^{q-1}) \equiv -\sqrt{c} \bmod qq'_{+-} \pmod{q}$; analogously for the other elements (Buchmann 2001, based on section 7.3.3).

3.9 Security: Rabin cipher security

3.25. The security of the Rabin cipher C informally means that the problem of decrypting some ciphertext c (in binary notation), computed by C, without knowing some private key (q, q') of c but knowing some public key qq' (in binary notation) of c, has no easy solution (that always works), when by an easy solution we understand a polynomial-time probabilistic algorithm that computes some $\sqrt{c} \bmod qq'_{\pm\mp}$ (with probability 1).

3.26 Definition. The Rabin cipher is *secure* if and only if

$$\forall A\ \exists qq', c\ \Pr[A((qq')_2, c_2) \in (\sqrt{c} \bmod qq')_2] < 1, \tag{3.1}$$

where

1. A ranges over the polynomial-time probabilistic algorithms;
2. qq' ranges over the Blum integers;
3. c ranges over the quadratic residues modulo qq'.

3.27. The security notion of the Rabin cipher seems ad hoc for this cipher because this security notion is about computing square roots modulo qq' and so it is the decryption function of this cipher; for other ciphers that have nothing to do with computing square roots modulo qq', this security notion does not fit. We argue now that this security notion can be recast as an instance of a more general security notion and in this sense it is less ad hoc than it seems. To do so, we first need to discuss asymmetric ciphers and their security, but for brevity we only sketch them.

Asymmetric cipher An *asymmetric cipher* is a 7-tuple $C = (\mathcal{K}, \mathcal{K}', \mathcal{P}, \mathcal{C}, K, E, D)$ where

1. $\mathcal{K}$, $\mathcal{K}'$, $\mathcal{P}$ and $\mathcal{C}$ are respectively *private-key*, *public-key*, *plaintext* and *ciphertext spaces*;
2. K, E and D are respectively a probabilistic *key-generation algorithm*, an *encryption* and a *decryption functions*, all polynomial time;
3. $K(1^n)$ outputs a *private-public pair of keys* $(k, k') \in \mathcal{K} \times \mathcal{K}'$;
4. C has *perfect* or *imperfect decryption* if and only if $D(k', \cdot)$ outputs respectively single plaintexts or sets/tuples of candidate plaintexts.
5. $D(k, \cdot)$ undoes $E(k', \cdot)$ in the sense of $\forall p \in \mathcal{P}\ p \overset{=}{\in} D(k, E(k', p))$ where we read "=" ("∈") if C has perfect (respectively, imperfect) decryption.

Motivation The goal of an asymmetric cipher is allow Bob to send encrypted messages to Alice, through a channel eavesdropped by Eve, without having previously arranged for a shared key:

1. Alice keeps k private and publishes k';
2. Bob encrypts p with k' and sends $c := E(k', p)$ to Alice;
3. Alice decrypts c with k to get $p \overset{=}{\in} D(k, c)$;
4. Eve cannot decrypt c because she does not know k.

Security An asymmetric cipher is ε*-secure* if and only if

$$\forall A\ \exists (k, k') \in \mathcal{K} \times \mathcal{K}'\ \exists c \in \mathcal{C}\ \ \Pr[A(k', c) \overset{=}{\in} D(k, c)] < \varepsilon, \tag{3.2}$$

where A ranges over the polynomial-time probabilistic algorithms, and $\varepsilon > 0$.

Instantiation Taking

1. $\varepsilon := 1$;
2. $(k, k') := ((q, q'), qq')$;
3. $D(k, c) := \sqrt{c} \bmod k'$, that is $D((q, q'), c) := \sqrt{c} \bmod qq'$;

in (3.2), we get (3.1) modulo conversions from $n \in \mathbb{N}$ to $n_2 \in \{0, 1\}^*$.

3.10 Primitive: binary-string function

3.28. A binary-string function is informally a function that maps strings to strings.

3.29 Definition.

1. A (total) *binary-string function* is a function $f\colon \{0,1\}^* \to \{0,1\}^*$.
2. A *partial binary-string function* is a function $f\colon \bigcup_{l\in L}\{0,1\}^l \to \{0,1\}^*$ where $L \subseteq \mathbb{N}$ is infinite.

3.11 Security: "collision resistance"

3.30. The "collision resistance" of a binary-string function f informally means that (f is polynomial-time computable and) it is hard to find collisions for f, that is ordered pairs $(x_1, x_2) \in \{0,1\}^* \times \{0,1\}^*$ such that $x_1 \neq x_2 \wedge f(x_1) = f(x_2)$.

3.31 "Definition". A binary-string function f is *"collision resistant"* if and only if:

1. f is polynomial-time computable;
2. we have

$$\forall A \ \Pr[A(1^n)_1 \neq A(1^n)_2 \wedge f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N},$$

   where A ranges over the polynomial-time probabilistic algorithms.

3.32. We write "collision resistance" inside quotation marks because this is not yet our official definition since it turns out to be an unwise definition as it is equivalent to injectivity.

3.33. The role of the input 1^n of A is not very clear in the "definition" above (for example if (x_1, x_2) is a collision for f, then the constant algorithm $A(x) := (x_1, x_2)$ outputs the collision (x_1, x_2) while ignoring the input $x = 1^n$) but only when we give our official definition below, however, for consistency and to help bridging the "definition" above and the definition below, we include the input already in the "definition" above.

3.12 Security: collision resistance

3.34. The collision resistance of a binary-string function f informally means that (f is polynomial-time computable and) it is hard to find arbitrarily long (or equivalently, infinitely many) collisions for f, that is ordered pairs $(x_1, x_2) \in \{0,1\}^* \times \{0,1\}^*$ such that $|x_1 x_2|$ is arbitrarily large and $x_1 \neq x_2 \wedge f(x_1) = f(x_2)$.

3.35 Definition. A binary-string function f is *collision resistant* if and only if:

1. f is polynomial-time computable;

2. we have

$$\forall A \;\; \Pr[|A(1^n)| \geq n \wedge A(1^n)_1 \neq A(1^n)_2 \wedge f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N},$$

where A ranges over the polynomial-time probabilistic algorithms.

3.36. We required the condition $|A(1^n)| \geq n$, that is the collisions $(A(1^n)_1, A(1^n)_2)$ have lengths $|A(1^n)| = |(A(1^n)_1, A(1^n)_2)| = |A(1^n)_1| + |A(1^n)_2|$ arbitrarily long, because without this condition the definition becomes equivalent to the injectivity of f.

3.37. The notion of collision resistance appears in the literature defined for a *family* of functions instead of a *single* function as in definition 3.35.

In previous work (Gaspar and Boiten 2014) we investigated composition theorems for one-way binary-string functions with the format

> if f (or g) is a one-way binary-string function and g (respectively f) is "special", then $f \circ g$ is a one-way binary-string function.

Collision resistance appeared as one of the "special" properties. Because the problem deals with single functions f and g, it forced us to formulate the notion of collision-resistant for a single function instead of a family of functions.

Let us present (a simplified variant of) the notion of collision resistance for a family of functions and then speculate about a relation with the notion of collision resistance for a single function.

1. A *family* of binary-string functions $F = \{f_n\}_{n \in \mathbb{N}}$ is collision resistant (Damgård 1988, definition 2.1) if and only if:
   (a) F is uniformly polynomial-time computable (that is there is a polynomial-time deterministic algorithm B such that $\forall n \in \mathbb{N} \, \forall x \in \{0,1\}^* \; B(1^n, x) = f_n(x)$);
   (b) we have
   $$\forall A \;\; \Pr[A(1^n)_1 \neq A(1^n)_2 \wedge f_n(A(1^n)_1) = f_n(A(1^n)_2)] \in \mathcal{N},$$
   where A ranges over the polynomial-time probabilistic algorithms.
2. Let us assume that there are functions
   $$\pi \colon \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*, \qquad \pi_1 \colon \{0,1\}^* \to \mathbb{N}, \qquad \pi_2 \colon \{0,1\}^* \to \{0,1\}^*$$
   that are respectively a bijective polynomial-time computable pairing function such that
   $$\forall n \in \mathbb{N} \, \forall x \in \{0,1\}^* \; n \leq |\pi(n,x)|$$
   and its two polynomial-time computable projections such that
   $$\forall n \in \mathbb{N} \, \forall x \in \{0,1\}^* \; \big(\pi_1(\pi(n,x)) = n \wedge \pi_2(\pi(n,x)) = x\big).$$

3. We speculate that the notions of collision resistant for *families* of binary-string functions and for *single* binary-string functions can be related somewhat along the lines of the following:

   (a) for all *families* $F = \{f_n\}_{n\in\mathbb{N}}$ of binary-string functions, if the *single* binary-string function f defined by $f(x) := f_{\pi_1(x)}(\pi_2(x))$ is collision resistant, then F is collision resistant (proof sketch: if (x, x') is a collision for f_n, then $(\pi(n,x), \pi(n,x'))$ is a collision for f of length at least $2n$);

   (b) for all *single* binary-string functions f, if the *family* $F = \{f_n\}_{n\in\mathbb{N}}$ of binary-string functions defined by $\forall n \in \mathbb{N}\ f_n := f$ is collision resistant, then f is collision resistant (proof sketch: a collision for f is a collision for f_n).

These ideas are still ongoing research.

3.13 Security: one-wayness

3.38. The one-wayness of a binary-string function f informally means that

1. f is easy to compute, that is $x \mapsto f(x)$ is easy to compute;
2. but f is hard to invert, that is $f(x) \mapsto x$, or more precisely, $f(x) \mapsto x'$ with $f(x) = f(x')$, is hard to compute.

3.39 Definition. A (possibly partial) binary-string function f is *one-way* (Goldreich 2004, definition 2.2.1) if and only if:

1. f is polynomial-time computable;
2. we have
$$\forall A \ \Pr\big[f\big(A(f(U_n), 1^n)\big) = f(U_n)\downarrow\big] \in \mathcal{N},$$
   where A ranges over the polynomial-time probabilistic algorithms.

To simplify, we often omit $\downarrow$.

3.40. In the definition above, the algorithm A, besides the input $f(U_n)$, also has the extra input 1^n. Informally, this extra input is included so that the runtime of A is not based on $|f(U_n)|$ but on $|f(U_n)1^n| = |f(U_n)| + n \geq n = |U_n|$, and in this way excludes functions f that are one-way only because they shrink U_n logarithmically into $f(U_n)$ and so any polynomial-time algorithm A trying to invert $f(U_n)$ would have to expand $f(U_n)$ exponentially into $A(f(U_n))$, which cannot be done in polynomial-time because there is not enough time for A to write down the expanded exponentially long $A(f(U_n))$ (Goldreich 2004, paragraph "The Auxiliary Input 1^n" in chapter 2).

3.41. In the definition above, we will always require the condition $f(U_n)\downarrow$ but often not write $\downarrow$. The condition makes no difference for total functions but may make a difference for partial functions (proof sketch: if

1. f is a total one-way binary string function;

2. g is the polynomial-time computable partial binary-string function $f|_{\{0,1\}^{2*}}$, that is f restricted to even-length strings;

3. $P_\downarrow := \Pr\big[g\big(A(g(U_n), 1^n)\big) = g(U_n)\downarrow\big]$ and $P := \Pr\big[g\big(A(g(U_n), 1^n)\big) = g(U_n)\big]$;

then g satisfies

1. $\forall A\ P_\downarrow \in \mathcal{N}$ by the one-wayness of f and because if n is odd, then $P_\downarrow = 0$ since $g(U_n)\uparrow$;

2. $\neg\forall A\ P \in \mathcal{N}$ by $A(x, y) := 1$ and because if n is odd, then $P = 1$ since $g(1) = g(U_n)$ due to $g(1)\uparrow$ and $g(U_n)\uparrow$.)

3.14 Primitive: pseudorandom generator

3.42. A pseudorandom generator is informally a function G that expands a short random string s into a “potentially infinitely long” pseudorandom string s_∞, in the sense of $\forall l \in \mathbb{N}\ G(s, 1^l) = s_\infty|_l$.

3.43 Definition. A *pseudorandom generator* (Goldreich 2004, definition 3.3.4) is a function $G\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that

1. G is polynomial-time computable;

2. $\forall s, l\ |G(s, 1^l)| = l$;

3. $\forall s, l\ G(s, 1^l) \sqsubseteq G(s, 1^{l+1})$;

where

1. s is called *seed* and ranges over $\{0,1\}^*$;

2. l is called *stream length* and ranges over $\mathbb{N}$;

3. $G(s, 1^l)$ is called *stream with seed s and of length l.*

3.15 Security: cryptographic security

3.44. The cryptographic security of a pseudorandom generator G informally means that it is hard for an algorithm A' to distinguish the stream $G(U_n, 1^{|A(1^n)_1|})$ (for a uniformly random seed U_n and of length $|A(1^n)_1|$ given by the first component $A(1^n)_1$ of an algorithm A) from a truly uniformly random string $U_{|G(U_n, 1^{|A(1^n)_1|})|}$ (of the same length as the stream).

3.45 Definition. A pseudorandom generator G is *cryptographically secure* (Goldreich 2004, adapted from definition 3.3.4) if and only if

$$\begin{aligned} \forall A, A'\ \ &\Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \\ &\Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N}, \end{aligned} \tag{3.3}$$

where A and A' range over the polynomial-time probabilistic algorithms.

3.46. Our formalisation above has a feature by design that we did not find in the literature: the first algorithm A is able to communicate with the second algorithm A' by means of the second component $A(\ldots)_2$ of A, which is passed as an input to A' in $A'(\ldots, A(\ldots)_2)$. This makes intuitive sense if we think of cryptographic security as asserting that the following two-step game is almost always unwinnable for a player (A, A'):

1. player (A, A'), through its first component A, chooses a length $l := |A(1^n)_1|$, for which player (A, A') hopes to distinguish between the pseudorandom string $G(U_n, 1^l) = G(U_n, 1^{|A(1^n)_1|})$ of that length l and the truly random string $U_l = U_{|G(U_n, 1^{|A(1^n)_1|})|}$ also of that length l;

2. player (A, A'), through its second component A', tries to distinguish between $G(U_n, 1^l)$ and U_l where l is the length chosen by player (A, A') in the previous game step.

Naturally, we want that player (A, A') in the second step still remembers what he/she did in the first step besides the chosen length l, for example player (A, A') may remember why he/she choose that particular value of l. This is formally achieved by allowing the first component A of player (A, A') to pass on an arbitrary piece of information $A(\ldots)_2$ to the second component A' of player (A, A') in the form of an input $A(\ldots)_2$ in $A'(\ldots, A(\ldots)_2)$. (One natural piece of information for A to pass to A' would be the entire computation history of A to assure that A' knows everything that A knows, but it would suffice for A' to pass on the entire list of "internal coin tosses" of A used in the computation made by A because A' can then recompute the entire computation history of A.) (Occasionally, it turns out that this communication from A to A' is not essential as we will see below when we remark that our definition of cryptographic security with communication is equivalent to Oded Goldreich's definition without communication, but for consistency, flexibility and ease of comparison our definitions always include communication.)

3.47. Oded Goldreich's book (Goldreich 2004, Goldreich 2011) is one of our main references, so let us us make a comparison with it. This book defines cryptographic security (Goldreich 2004, definition 3.3.4) essentially by a formula seemingly weaker than (3.3), namely

$$\forall p, B' \ \Pr B'(G(U_n, 1^{p(n)}), 1^n) - \Pr B'(U_{|G(U_n, 1^{p(n)})|}, 1^n) \in \mathcal{N}, \tag{3.4}$$

where

1. p ranges over the positive polynomials;

2. B' ranges over the polynomial-time probabilistic algorithms;

but it turns out that (3.3) and (3.4) are equivalent (proof sketch:

<u>$(\Rightarrow)$</u> taking $A(x) := (1^{p(|x|)}, \epsilon)$ and $A'(x, y, z) := B'(x, y)$ in (3.3), we get (3.4);

$\underline{(\Leftarrow)}$ for all polynomial-time probabilistic algorithms A, there is a positive polynomial p_A such that $\forall n \in \mathbb{N}\ |A(1^n)_1| \leq p_A(n)$, so taking $p := p_A$ and $B'(x, y) := A'(x|_{|A(y)_1|}, y, A(y)_2)$ in (3.4), we get

$$\forall A, A' \ \Pr A'(\overbrace{G(U_n, 1^{p_A(n)})|_{|A(1^n)_1|}}^{=G(U_n,1^{|A(1^n)_1|})}, 1^n, A(1^n)_2) -$$
$$\Pr A'(\underbrace{U_{|G(U_n,1^{p_A(n)})|}|_{|A(1^n)_1|}}_{=U_{|G(U_n,1^{|A(1^n)_1|})|}}, 1^n, A(1^n)_2) \in \mathcal{N},$$

thus we get (3.3)).

3.16 Primitive: stream cipher

3.48. A stream cipher is informally a cipher that encrypts a "potentially infinitely long" plaintext p_∞ to a "potentially infinitely long" ciphertext c_∞ in the sense that a prefix p of p_∞ is encrypted to a prefix c of c_∞, or somewhat more precisely, as we add extra bits to a plaintext p getting a longer plaintext $p' \sqsupseteq p$, the stream cipher adds extra bits to the ciphertext c producing a longer ciphertext $c' \sqsupseteq c$.

3.49 Definition. A (symmetric and deterministic) *stream cipher* C is a 6-tuple $(\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ where

1. $\mathcal{K} := \{0, 1\}^*$ is called *key space*;
2. $\mathcal{P} := \{0, 1\}^*$ is called *plaintext space*;
3. $\mathcal{C} := \{0, 1\}^*$ is called *ciphertext space*;
4. K a is probabilistic polynomial-time algorithm called *key generator* such that $\forall x \in \{0, 1\}^*\ K(x) \in \mathcal{K}$;
5. $E\colon \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ is a polynomial-time computable function called *encryption function*.
6. $D\colon \mathcal{K} \times \mathcal{C} \to \mathcal{P}$ is a polynomial-time computable function called *decryption function*;

such that

1. $\forall k \in \mathcal{K}\ \forall p \in \mathcal{P}\ D(k, E(k, p)) = p$;
2. $\forall k \in \mathcal{K}\ \forall p, p' \in \mathcal{P}\ (p \sqsubseteq p' \ \Rightarrow\ E(k, p) \sqsubseteq E(k, p'))$.

3.50. Our definition of stream cipher is restricted to stream ciphers that are:

1. symmetric (that is ciphers in which the encryption and decryption keys are the same or at least the latter can be easily computed from the former), so excluding asymmetric ciphers (for example the Blum-Blum-Shub stream cipher (Klein 2013, chapter 11));

2. deterministic (that is the encryption is computed by a deterministic algorithm instead of a probabilistic algorithm), so excluding enhancements on stream ciphers by adding randomness to them (for example random nonces and random initialisation vectors) and casting some unavoidable insecurity (for example if two ciphertexts under the same key are seen to be equal, then it can be inferred that the corresponding plaintexts are also equal).

3.17 Security: indistinguishability from random

3.51. The indistinguishability from random of a stream cipher C informally means that it is hard for an algorithm A' to distinguish a ciphertext $E(K(1^n), A(1^n)_1)$ computed by C (with corresponding plaintext given by the first component $A(1^n)_1$ of an algorithm A) from a uniformly random string $U_{|E(K(1^n),A(1^n)_1)|}$ (of the same length as the ciphertext).

3.52 Definition. A stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ is *indistinguishable from random* (Möller 2004, paragraph before definition 3) (Reingold 1998, section 1.1.1) (Rogaway 2004, section 3) if and only if

$$\forall A, A' \ \Pr A'\big(E(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\big) - \Pr A'(U_{|E(K(1^n),A(1^n)_1)|}, 1^n, A(1^n)_2) \in \mathcal{N},$$

where A and A' range over the polynomial-time probabilistic algorithms.

3.53. In the definition, the last component $A(\ldots)_2$ of A plays the same role (of passing extra information from A to A') as the one in paragraph 3.46. This design choice is standard for us (but not for others because we did not find it in the literature), so we will often adopt it without commenting on it.

3.18 Security: indistinguishable encryptions

3.54. The indistinguishability of encryptions of a stream cipher C informally means that it is hard for an algorithm A' to distinguish between two ciphertexts $E(K(1^n), A(1^n)_1)$ and $E(K(1^n), A(1^n)_2)$ computed by C (with corresponding plaintexts of the same length and given by the first and second components $A(1^n)_1$ and $A(1^n)_2$ of an algorithm A).

3.55 Definition. A stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ has *indistinguishable encryptions* (Katz and Lindell 2015, adapted from definition 3.8) if and only if

$$\forall A, A' \ \Pr A'\big(E(K(1^n), A(1^n)_1), A(1^n)_1, A(1^n)_2, 1^n, A(1^n)_3\big) - \Pr A'\big(E(K(1^n), A(1^n)_2), A(1^n)_1, A(1^n)_2, 1^n, A(1^n)_3\big) \in \mathcal{N},$$

where A and A' range over the polynomial-time probabilistic algorithms such that for A we have $\forall x \in \{0,1\}^* \ |A(x)_1| = |A(x)_2|$.

3.56. As before, the last component $A(\ldots)_3$ of A plays the same role as the one of $A(\ldots)_2$ in paragraph 3.46.

3.19 Security: semantic security

3.57. The semantic security of a stream cipher C informally means that ciphertexts computed by C do not reveal any computationally-extractable information about their corresponding plaintexts, in the sense that what can be computed from seeing a ciphertext can also be computed without seeing the ciphertext, or more precisely, that a function g computable by an algorithm A' from a ciphertext $E(K(1^n), A(1^n)_1)$ (with corresponding plaintext given by the first component $A(1^n)_1$ of an algorithm A) (and also from extra information about the plaintext given by a function f) can also be computed by an algorithm B without access to $E(K(1^n), A(1^n)_1)$.

3.58 Definition. A stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ is *semantically secure* (Katz and Lindell 2015, based on definition 3.12) (Goldreich 2011, based on definition 5.2.1) if and only if

$$\begin{gathered}\forall A'\ \exists B\ \forall A, f, g \\ \Pr\big[A'\big(E(K(1^n), A(1^n)_1), f(A(1^n), 1^n), 1^n, A(1^n)_2\big) = g(A(1^n), 1^n)\big] - \\ \Pr\big[B\big(1^{|A(1^n)_1|}, f(A(1^n), 1^n), 1^n, A(1^n)_2\big) = g(A(1^n), 1^n)\big] \in \mathcal{N},\end{gathered}$$

where

1. A, A' and B range over the polynomial-time probabilistic algorithms;

2. f and g range over the polynomial-time computable functions.

3.20 Security: bit-recovery resistance

3.59. The bit-recovery resistance of a stream cipher C informally means that it is hard for an algorithm A' to recover (with probability better than the probability 1/2 of a uniformly random guess) the i-th bit $U^i_{|A(1^n)_1|}$ of a uniformly random plaintext $U_{|A(1^n)_1|}$ (of length $|A(1^n)_1|$ given by the first component $A(1^n)_1$ of an algorithm A) from the corresponding ciphertext $E(K(1^n), U_{|A(1^n)_1|})$ computed by C.

3.60 Definition. A stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ is *bit-recovery resistant* (Katz and Lindell 2015, based on theorem 3.10) if and only if

$$\forall A, A', i\ \Pr\big[A'\big(E(K(1^n), U_{|A(1^n)_1|}), 1^n, A(1^n)_2\big) = U^i_{|A(1^n)_1|}\big] - 1/2 \in \mathcal{N},$$

where

1. A and A' range over the polynomial-time probabilistic algorithms such that for A we have $\forall x \in \{0,1\}^*\ |A(x)_1| \geq 1$;

2. i ranges over $\mathbb{N}$.

3.21 Primitive: formal language

3.61. A formal language is informally a set of words written with letters taken from some alphabet.

3.62 Definition. Let Σ be a set.

1. The set Σ is called an *alphabet* (Papadimitriou 1994, definition 2.1).
2. The elements of Σ are called *letters over* Σ.
3. The finite strings (that is n-tuples with $n \in \mathbb{N}$) $w = l_1 \dots l_n$ of letters over Σ are called *words over* Σ.
4. The sets L whose elements are words over Σ are called *languages over* Σ (Papadimitriou 1994, definition 2.3).

3.22 Security: $(\mathrm{NP} \setminus \mathrm{P})$-ness

3.63. Let c be a constant and A an algorithm. We will use the notations $\Pr[A(w) = c] > 0$ and $\Pr[A(w) = c] = 1$, which are

1. defined if A is probabilistic (because there is a notion of probability for A);
2. undefined if A is deterministic or nondeterministic (because there is no notion of probability for A);

so we need to clarify the meaning of the notations in the latter case:

1. $\Pr[A(w) = c] > 0$ means that it is possible for $A(w)$ to output c, that is there is a computation path of A starting on input w and ending on output c;
2. $\Pr[A(w) = c] = 1$ means that it is certain that $A(w)$ outputs c, that is all computation paths of A starting on input w end on output c.

3.64. The $(\mathrm{NP} \setminus \mathrm{P})$-ness of a formal language L informally means that the problem of answering "yes" or "no" to the question "$w \in L$?" is hard in the sense of being solvable by

1. a polynomial-time nondeterministic algorithm;
2. but not a polynomial-time deterministic algorithm.

3.65 Definition. Let L be a formal language over an alphabet Σ, and `yes` and `no` be two constants.

1. Let A be a polynomial-time (deterministic, nondeterministic or probabilistic) algorithm that inputs words w over Σ and outputs `yes` or `no`. The algorithm A *decides* L if and only if

$$\begin{aligned} \forall w \ \big((w \in L \ &\Rightarrow \ \Pr[A(w) = \mathtt{yes}] > 0) \ \wedge \\ (w \notin L \ &\Rightarrow \ \Pr[A(w) = \mathtt{no}] \ \ = 1) \big), \end{aligned} \tag{3.5}$$

where w ranges over the words over Σ (Papadimitriou 1994, definition 2.3).

2. The formal language L is:

   (a) P if and only if there is a polynomial-time deterministic algorithm deciding L;

   (b) NP if and only if there is a polynomial-time nondeterministic algorithm deciding L;

   (c) $\mathrm{NP} \setminus \mathrm{P}$ if and only if L is NP but not P (Papadimitriou 1994, sections 2.3 and 2.7).

3.23 Conclusion

3.66. In this chapter we presented a collection of formalisations of cryptographic primitives and security notions.

Part III

Examples

Chapter 4

Provable security of transformation of cryptographic primitives

4.1 Introduction

4.1. In this chapter, we present two classic examples of provable security, adapted to fit our problem format of transformation of cryptographic primitives:

1. the transformation of a uniform random generator into a perfectly-secret one-time pad;
2. the transformation of the hard Blum-integer factorisation problem into the secure Rabin cipher.

4.2. In this chapter, we give two contributions modestly worth of notice:

1. a proof of the perfect secrecy of the one-time pad that is direct and does not use Bayes' theorem $\Pr[A|B] = \Pr[B|A]\ \ \Pr A \ / \ \Pr B$ (if $\Pr A, \Pr B \neq 0$);
2. a more formalised and yet more readable proof (than the ones that we found in the literature) of the security of the Rabin cipher.

The proof of the perfect secrecy of the one-time pad that we present has two advantages when compared to some other proofs found in the literature.

Bayes-free Our proof avoids using Bayes' theorem for simplicity. This is already done in some proofs in the literature

(Boneh and Shoup 2017, proofs of theorems 2.1, 2.2 and 2.4)
(Delfs and Knebl 2007, proofs of proposition 9.4 and of theorem 9.5)
(Pass and Shelat 2010, proofs of theorem 12.3 and of proposition 15.5)
(Talbot and Welsh 2006, proof of proposition 5.5)
(Vaudenay 2006, proof of theorem 1.5)

but not in some other proofs in the literature

(Blahut 2014, proofs of theorem 5.3.2 and of corollary 5.3.3)
(Buchmann 2001, proof of theorem 4.4.3, and second paragraph of section 4.5)
(Katz and Lindell 2015, proofs of lemma 2.4 and of theorem 2.9, and paragraph before theorem 2.9)
(Rothe 2005, proof of theorem 4.24, and example 4.25)
(Shannon 1949, proof of theorem 6, and page 682)
(Smart 2016, proof of theorem 9.4).

Direct Our proof is direct in the sense that it avoids relying on auxiliary results for simplicity. Again, this is already done in some proofs in the literature but not in some other proofs in the literature, as we can verify by inspecting the list of references in the previous point in order to check if they cite a single proof (for example "proof of theorem 9.4") or multiple proofs (for example "proofs of theorems 2.1, 2.2 and 2.4").

4.2 Example: transformation of a uniform random generator into the perfectly-secret one-time pad

4.3. Let us see an adaptation to fit our problem format (presented in paragraph 1.28) of a classical result (the perfect secrecy of the one-time pad) specified by the following instances of its parameters:

$$\begin{aligned}
P_1 &= n\text{-length random generator,} \\
S_1 &= n\text{-length uniformity,} \\
P_2 &= n\text{-length one-time pad,} \\
S_2 &= n\text{-length perfect secrecy,} \\
T &= \text{transformation of an } n\text{-length random} \\
&\quad\ \text{generator into an } n\text{-length one-time pad.}
\end{aligned}$$

4.4. Let us start by sketching our example.

n-length random generator Informally, it is an algorithm G that outputs a binary string of length n such as 001001010 (where $n = 9$).

n-length uniformity Informally, it means that the binary strings of length n output by the n-length random generator are all equally probable.

n-length one-time pad Informally, it encrypts a plaintext p of length n with a key k of length n (produced by G) by xoring p and k getting $k \oplus p$, and decrypts by doing the same.

n-length perfect secrecy Informally, it means that a ciphertext of length n reveals no information about the plaintext (in the sense of singling out a plaintext as more probable than the other ones).

Security theorem We prove a theorem saying that if the n-length random generator is n-length uniform, then the n-length one-time pad is n-length perfectly secret.

The idea of the proof is the following: a ciphertext c can be the result of encrypting any plaintext p with the key $k := p \oplus c$ where each key k is equally probable (by the n-length uniformity of G), so each plaintext p is equally probable, thus c does not single out any p.

4.5. Let us introduce our transformation T: the transformation of an n-length random generator into an n-length one-time pad.

4.6 Definition. Let $n \in \mathbb{N}$. The transformation of an n-length random generator G_n into an n-length one-time pad C_{n,G_n} is $G_n \rightsquigarrow C_{n,G_n}$.

4.7. Finally, let us present our security theorem.

4.8 Theorem. Let $n \in \mathbb{N}$. For all n-length random generators G_n, if G_n is n-length uniform, then C_{n,G_n} is n-length perfectly secret (Buchmann 2001, theorem 4.4.3).

4.9 Proof. The definition of $\Pr[p|c]$ is

$$\Pr[p|c] := \frac{\Pr_{\mathcal{K}_n \times \mathcal{P}_n} P_p \cap C_c}{\Pr_{\mathcal{K}_n \times \mathcal{P}_n} C_c}. \tag{4.1}$$

Using $E_n(k', p') = k' \oplus p'$ (by definition of E_n) we get

$$\begin{gathered}(k', p') \in C_c \;\Leftrightarrow\; E(k', p') = c \;\Leftrightarrow\\ k' \oplus p' = c \;\Leftrightarrow\; k' = c \ominus p' \;\Leftrightarrow\; (k', p') = (c \ominus p', p'),\\ (k', p') \in P_p \cap C_c \;\Leftrightarrow\; p' = p \wedge (k', p') = (c \ominus p', p') \;\Leftrightarrow\; (k', p') = (c \ominus p, p),\end{gathered}$$

so

$$C_c = \{(c \ominus p', p') \mid p' \in \mathcal{P}_n\}, \tag{4.2}$$

$$P_p \cap C_c = \{(c \ominus p, p)\} \tag{4.3}$$

(for all $p \in \mathcal{P}_n$ and $c \in \mathcal{C}_n$). The definition of $\Pr_{\mathcal{K}_n \times \mathcal{P}_n}$ is

$$\Pr_{\mathcal{K}_n \times \mathcal{P}_n}(k, p) := \Pr_{\mathcal{K}_n} k \times \Pr_{\mathcal{P}_n} p \tag{4.4}$$

(for all $p \in \mathcal{P}_n$ and $c \in \mathcal{C}_n$) because we assume that $\Pr_{\mathcal{K}_n}$ and $\Pr_{\mathcal{P}_n}$ are independent. The n-length random generator G_n being n-length uniform means $G_n(1^n) = U_n$, where $K_{n,G_n} := G_n$ and $\Pr_{\mathcal{K}_n} k := \Pr[K_{n,G_n}(1^n) = k]$, so

$$\Pr_{\mathcal{K}_n} k = 2^{-n} \tag{4.5}$$

(for all $k \in \mathcal{K}_n$). By the definition of probability (distribution), we have

$$\sum_{p' \in \mathcal{P}_n} \Pr_{\mathcal{P}_n} p' = 1. \tag{4.6}$$

Recall

$$\Pr p = \Pr_{\mathcal{P}_n} p. \tag{4.7}$$

The one-time pad being perfectly secret means

$$\Pr p = \Pr[p|c] \tag{4.8}$$

(for all $p \in \mathcal{P}_n$ and $c \in \mathcal{C}_n$).

Using (4.1), (4.2), (4.3), (4.4), (4.5) and (4.6) where indicated, we have

$$\begin{aligned}
\Pr[p|c] = & \qquad \text{by (4.1)} \\
\frac{\Pr_{\mathcal{K}_n \times \mathcal{P}_n} P_p \cap C_c}{\Pr_{\mathcal{K}_n \times \mathcal{P}_n} C_c} = & \qquad \text{by (4.2)–(4.3)} \\
\frac{\Pr_{\mathcal{K}_n \times \mathcal{P}_n} (c \ominus p, p)}{\sum_{p' \in \mathcal{P}_n} \Pr_{\mathcal{K}_n \times \mathcal{P}_n} (c \ominus p', p')} = & \qquad \text{by (4.4)} \\
\frac{\Pr_{\mathcal{K}_n} c \ominus p \times \Pr_{\mathcal{P}_n} p}{\sum_{p' \in \mathcal{P}_n} \Pr_{\mathcal{K}_n} c \ominus p' \times \Pr_{\mathcal{P}_n} p'} = & \qquad \text{by (4.5)} \\
\frac{2^{-n} \Pr_{\mathcal{P}_n} p}{2^{-n} \sum_{p' \in \mathcal{P}_n} \Pr_{\mathcal{P}_n} p'} = & \qquad \text{by (4.6)} \\
\Pr_{\mathcal{P}_n} p = & \qquad \text{by (4.7)} \\
\Pr p, &
\end{aligned}$$

so we get (4.8).

4.3 Example: transformation of the hard Blum-integer factorisation problem into the secure Rabin cipher

4.10. Let us see an adaptation to fit our problem format (presented in paragraph 1.28) of another classical result (the security of the Rabin cipher) specified by the following instances of its parameters:

P_1 = Blum-integer factorisation problem,
S_1 = hardness,
P_2 = Rabin cipher,
S_2 = security,
T = transformation of the Blum-integer factorisation problem into the Rabin cipher.

Actually, talking about a transformation T in this example is to some extent artificial: we are not going to build P_2 using P_1 but rather prove the S_2-security of P_2 using the S_1-security of P_1; it is in this broader sense that we talk of a transformation T. Along the same lines, considering the Blum-integer factorisation problem as a cryptographic primitive P_1 and its hardness as a cryptographic security notion S_1 is to some extent artificial: they are more related to number theory than to cryptography but they do play an essential role in the security of the Rabin cipher, which belongs to cryptography; it is in this broader sense that we talk about the cryptographic primitive P_1 and its security notion S_1.

4.11. Let us start by sketching our example.

Number theory We will need to recall how to calculate square roots modulo n and how to transform two square roots p and p' modulo n into a "magic number" $\mathrm{mn}(p, p')$ that gives us a factor of n.

Factorisation problem Informally, it is the problem of finding a (non-trivial) factor of any given n.

Hardness Informally, it means that it is hard to factor n.

Rabin cipher Informally, it encrypts a plaintext p by squaring p modulo n and decrypts a ciphertext c by calculating square roots of c modulo n.

Security Informally, it says that it is difficult to decrypt a ciphertext (without knowing the factorisation of n).

Security theorem We prove a theorem essentially saying that if the factorisation problem is hard, then the Rabin cipher is secure, or equivalently, if we know how to decrypt the Rabin cipher, that is how to calculate a square root modulo n, then we know how to factor n.

The idea of the proof is the following: we chose a random p, so we have a square root p of p^2 modulo n; since we know how to compute a square root modulo n, then we get another square root p' of p^2 modulo n; now we use $\mathrm{mn}(p, p')$ to get a factor of n.

4.12. Before we progress to the presentation of our example, we need to make a small digression into number theory, more precisely Blum primes, Blum integers, quadratic residues, square roots modulo n, the factorisation problem and the relation between square roots modulo n and the factorisation problem.

4.13. Let qq' be a Blum integer and let us define the *"magic number"* $\mathrm{mn}(p, p') := \gcd(p-p', qq')$. The "magic number" transforms two square roots p and p' modulo qq' of the same quadratic residue modulo qq' into a factor of qq' with probability at least $1/2$. More precisely, we make the claim (Buchmann 2001, section 7.3.5)

> if qq' is a Blum integer, c is a quadratic residue modulo qq', p is uniformly randomly chosen in $\sqrt{c} \bmod qq'$, p' is arbitrarily chosen in $\sqrt{c} \bmod qq'$ independently from p, and $\gcd(p, qq') = 1$, then $\Pr[\mathrm{mn}(p, p') \in \{q, q'\}] \geq 1/2$.

proved in proof 4.18. (We chose to call "magic number" to $\mathrm{mn}(p, p')$ because it seems to "magically" factor qq'.) The hypothesis $\gcd(p, qq') = 1$ may seem a little strange, it is here for a technical reason appearing in the proof 4.18, but does not "hurt" because if $\gcd(p, qq') \neq 1$, then $\gcd(p, qq') \in \{q, q'\}$ (because $0 \neq p \in \sqrt{c} \bmod qq' \subseteq [0 \mathinner{..} qq' - 1]$, so $p \in [1 \mathinner{..} qq' - 1]$), so we get a factor of qq' as intended.

4.14. Finally, let us present our security theorem and transformation T: the reduction of the security of the Rabin cipher to the hardness of the Blum-integer factorisation problem.

4.15 Theorem. If the Blum-integer factorisation problem is hard, then the Rabin cipher is secure (Buchmann 2001, section 7.3.5).

4.16 Proof. We prove the contrapositive of the theorem. The Rabin cipher not being secure means

$$\exists A\ \forall qq', c\ \ \Pr[A((qq')_2, c_2) \in (\sqrt{c} \bmod qq')_2] = 1, \tag{4.9}$$

where A ranges over the polynomial-time probabilistic algorithms, qq' ranges over the Blum integers and c ranges over the quadratic residues modulo qq'. The Blum-integer factorisation problem not being hard means

$$\exists B\ \forall qq'\ \ \Pr[B((qq')_2) \in \{q_2, q'_2\}] \geq 1/2, \tag{4.10}$$

where B ranges over the polynomial-time probabilistic algorithms and qq' ranges over the Blum integers.

Let us construct such a B from such an A. Let B be a polynomial-time probabilistic algorithm that:

1. inputs $(qq')_2$;
2. (a) generates a uniformly random number $p \in [0 \mathinner{..} qq' - 1]$;
   (b) computes $\gcd(p, qq')_2$;
   (c) computes $c := p^2 \bmod qq'$;
   (d) computes $p' := {}_2A((qq')_2, c_2)$;
   (e) computes $\mathrm{mn}(p, p')_2$;
3. outputs
   (e) $\gcd(p, qq')_2$ if $\gcd(p, qq') \neq 1$;
   (f) $\mathrm{mn}(p, p')_2$ if $\gcd(p, qq') = 1$;

which is a polynomial-time probabilistic algorithm because the generation of p can be done in polynomial time in $|(qq')_2|$, gcd, so also mn, is a polynomial-time computable function, and A is a polynomial-time probabilistic algorithm. (To be rigorous, we should have defined the behaviour of B on an general input $x \in \{0,1\}^*$ instead of a particular input $(qq')_2$, and then some small changes would be required such as where it is $(qq')_2$ should be x and where it is qq' should be ${}_2x$, but this extra rigour makes the proof more difficult to read so we skipped it.)

Let us prove $\Pr[B((qq')_2) \in \{q_2, q'_2\}] \geq 1/2$:

1. if the output $B((qq')_2)$ is $\gcd(p, qq')_2$ from point 3e, then $\Pr[B((qq')_2) \in \{q_2, q'_2\}] = 1$ (because $\gcd(p, qq') \in \{1, q, q', qq'\}$ since $q, q' \in \mathbb{P}$, and $1 < \gcd(p, qq') \leq p \leq qq' - 1 < qq'$ by the generation of p and by point 3e);

2. if the output $B((qq')_2)$ is $\mathrm{mn}(p,p')_2$ from point 3f, then by the claim in paragraph 4.13 we get $\Pr[B((qq')_2) \in \{q_2, q_2'\}] \geq 1/2$.

4.17. As promised, we give a proof of the claim made in paragraph 4.13.

4.18 Proof. First we make two claims.

1. Claim: p is a square root of c modulo qq' if and only if p is a square root of c modulo q and q'.

   Proof: given in proof 3.24.

2. Claim: two square roots p and p' of c modulo q differ at most in their signs modulo q; analogously for q' instead of q.

   Proof: given in proof 3.24.

We have one of the following 4 equally-probable cases (because p is uniformly random and p' is independent from p) using the previous claims:

$$p \equiv +p' \pmod q \quad \wedge \quad p \equiv +p' \pmod{q'}, \tag{4.11}$$
$$p \equiv +p' \pmod q \quad \wedge \quad p \equiv -p' \pmod{q'}, \tag{4.12}$$
$$p \equiv -p' \pmod q \quad \wedge \quad p \equiv +p' \pmod{q'}, \tag{4.13}$$
$$p \equiv -p' \pmod q \quad \wedge \quad p \equiv -p' \pmod{q'}. \tag{4.14}$$

We make three more claims.

3. Claim: we have $\mathrm{mn}(p,p') \in \{1, q, q', qq'\}$.

   Proof: we have $\mathrm{mn}(p,p') \mid qq'$ by definition of $\mathrm{mn}(p,p')$ and of gcd, and the only positive divisors of qq' are 1, q, q' and qq' because $q, q' \in \mathbb{P}$.

4. Claim: in cases (4.12) and (4.13) we have $\mathrm{mn}(p,p') \neq 1$.

   Proof: in case (4.12) we have $q \mid p-p'$, so $q \mid \mathrm{mn}(p,p')$, thus $\mathrm{mn}(p,qq') \geq q > 1$ since $q \in \mathbb{P}$; analogously for case (4.13).

5. Claim: in cases (4.12) and (4.13) we have $\mathrm{mn}(p,p') \neq qq'$.

   Proof: we have $p, p' \in \sqrt{c} \bmod qq' \subseteq [0 \mathinner{..} qq'-1]$, so $(*)$ $p-p' \in [-(qq'-1) \mathinner{..} qq'-1]$; if $p - p' = 0$, then $q' \mid 2p$ by (4.12), so $q' \mid p$ because $q' \in \mathbb{P} \setminus \{2\}$, thus $\gcd(p, qq') \geq q' > 1$, contradicting $\gcd(p, qq') = 1$, so $(\dagger)$ $p - p' \neq 0$; from $(*)$ and $(\dagger)$ we get the claim; analogously for case (4.13).

In cases (4.12) and (4.13) we have $\mathrm{mn}(p,p') \in \{q, q'\}$. So $\mathrm{mn}(p,p') \in \{q, q'\}$ in at least 2 out of 4 equally-probable cases, thus $\Pr[\mathrm{mn}(p,p') \in \{q,q'\}] \geq 1/2$ (Buchmann 2001, section 7.3.5).

4.4 Conclusion

4.19. In this chapter we presented two classic examples of provable security of transformation of cryptographic primitives:

1. the transformation of a uniform random generator into a perfectly-secret one-time pad;
2. the transformation of the hard Blum-integer factorisation problem into the secure Rabin cipher.

Chapter 5

Proof presentation of transformation of cryptographic primitives

5.1 Introduction

5.1. In this chapter we present three examples of proof presentation:

1. a bad cryptographic definition (a good-looking attempt to capture the notion of collision resistance that fails by turning out to be equivalent to the notion of injectivity) to illustrate the delicateness of this type of definitions;

2. an incorrect cryptographic proof (essentially, that if a function is collision resistant, then it is one-way) to illustrate the delicateness of this type of proofs;

3. an improvement of a cryptographic proof (that if we change an arbitrary one-way function f to force $\forall n \in \mathbb{N}\ f(0^n) = \epsilon$, then f is still one-way) by using the simple notation trick of replacing a long formula $\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ by a short abbreviation $f \in \mathcal{N}$ (together with some easy and useful properties of $\mathcal{N}$) to illustrate how to improve a proof.

5.2. The

1. notion of "collision resistance" (the wrong notion with quotation marks);

2. collision resistance (the right notion without quotation marks);

3. the result that "collision resistance" is equivalent to injectivity;

4. the idea of replacing $\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ by $f \in \mathcal{N}$;

5. the properties of $\mathcal{N}$;

have been developed in collaboration with Eerke Boiten (Gaspar and Boiten 2014, sections 1.3 and 2.5).

5.2 Example: collision resistance

5.3. Let us see an example showing how delicate definitions in cryptography can be. The example is a definition which seems to capture well an intuitive notion but turns out not to do it well.

5.4 "Definition". A binary-string function f is *"collision resistant"* if and only if:

1. f is polynomial-time computable;

2. we have

$$\forall A \ \Pr[A(1^n)_1 \neq A(1^n)_2 \ \wedge \ f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N}, \tag{5.1}$$

   where A ranges over the polynomial-time probabilistic algorithms.

5.5. The "definition" above seems to capture well the intuitive notion of being hard to find collisions for f by using the following standard techniques in cryptography:

1. modelling an adversary trying to find a collision by a polynomial-time probabilistic algorithm A in the part "$\forall A$" of (5.1);

2. having A attempt to compute a collision $(A(1^n)_1, A(1^n)_2)$ for f in the part "$A(1^n)_1 \neq A(1^n)_2 \ \wedge \ f(A(1^n)_1) = f(A(1^n)_2)$" of (5.1);

3. requiring that A only succeeds with negligible probability in the part "$\Pr[\ldots] \in \mathcal{N}$" of (5.1).

However, the proposition below shows that the "definition" reduces to the injectivity of f, which is not the notion that the "definition" intends to capture. So the "definition" is an example of a "definition" that seems good (that is capturing the intended intuitive notion) but turns out to be bad (that is not capturing the intended intuitive notion). As such, the "definition" exemplifies our thesis that it is easy to make mistakes but hard to find them in definitions in cryptography.

5.6 Proposition. For all binary-string functions f, we have: f is *"collision resistant"* if and only if f is injective.

5.7 Proof.

$\underline{\Rightarrow}$ We prove the contrapositive of the proposition. If f is not injective, that is there are $x_1, x_2 \in \{0,1\}^*$ such that $x_1 \neq x_2$ and $f(x_1) = f(x_2)$, then taking the constant algorithm $A(x) := (x_1, x_2)$ (which is a polynomial-time, even constant-time, probabilistic, even deterministic, algorithm) we have $A(1^n)_1 \neq A(1^n)_2 \wedge f(A(1^n)_1) = f(A(1^n)_2)$, so

$$\Pr[A(1^n)_1 \neq A(1^n)_2 \ \wedge \ f(A(1^n)_1) = f(A(1^n)_2)] = 1 \notin \mathcal{N},$$

thus f is not "collision resistant".

$\underline{\Leftarrow}$ If f is injective, then f has no collisions, so $B(1^n)_1 \neq B(1^n)_2 \wedge f(B(1^n)_1) = f(B(1^n)_2)$ is false for all B, thus

$$\forall B \ \Pr[B(1^n)_1 \neq B(1^n)_2 \wedge f(B(1^n)_1) = f(B(1^n)_2)] = 0 \in \mathcal{N},$$

hence f is "collision resistant".

5.8. A closer look at the proof above shows that the essence of why the notion of "collision resistant" reduces to injectivity is that the algorithm A is only challenged to produce a single collision (x_1, x_2), which it can by having the collision "hard-wired" in it instead of spending computational effort finding the collision, and the same would work if the algorithm were only challenged to produce finitely many collisions. The "correction" of the definition of "collision resistance" that we proposed avoid this problem by challenging the algorithm to produce infinitely many collisions, or equivalent, arbitrarily long collisions (that is $|x_1|$ or $|x_2|$ arbitrarily long, or equivalently, $|x_1x_2| = |x_1| + |x_2|$ arbitrarily long).

5.9 Definition. A binary-string function f is *collision resistant* if and only if:

1. f is polynomial-time computable;

2. we have

   $$\forall A \ \Pr[|A(1^n)| \geq n \wedge A(1^n)_1 \neq A(1^n)_2 \wedge f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N},$$

   where A ranges over the polynomial-time probabilistic algorithms.

5.3 Example: two-to-one collision resistance implies one-wayness

5.10. Let us see an example showing how delicate proofs in cryptography can be. The example is a "proof" that seems to be correct but turns out to contain a subtle mistake.

5.11 "Proposition". If a binary-string function f is two-to-one and collision resistant, then f is one-way (Canetti 2009, section 1.1 in lecture 7).

5.12 "Proof". We prove essentially the transposition of the proposition. If f is not one-way but it is polynomial-time computable, that is there is a polynomial-time probabilistic algorithm A such that $P(n) := \Pr\big[f\big(A(f(U_n), 1^n)\big) = f(U_n)\big] \notin \mathcal{N}$, then the polynomial-time probabilistic algorithm B defined by $B(x) := \big(U_{|x|}, A(f(U_{|x|}), 1^{|x|})\big)$ satisfies the following three properties.

$\underline{|B(1^n)| \geq n}$ Follows from $|B(1^n)| \geq |B(1^n)_1|$, $B(1^n)_1 = U_n$ and $|U_n| = n$.

$\underline{\overbrace{f(B(1^n)_1) = f(B(1^n)_2)}^{(\dagger)} \Rightarrow \overbrace{\Pr[B(1^n)_1 \neq B(1^n)_2] = 1/2}^{(*)}}$ If $(\dagger)$, then $B(1^n)_1$ and $B(1^n)_2$ are preimages of the same image y, there are exactly two such preimages since f is two-to-one, one of the preimages is $B(1^n)_2$ and $B(1^n)_1$ has probability $1/2$ of being the other preimage because $B(1^n)_1$ is uniformly random, so $(*)$.

$\underline{\Pr[f(B(1^n)_1) = f(B(1^n)_2)] = P(n)}$ We have that $f(B(1^n)_1) = f(B(1^n)_2)$ is $f(U_n) = f\big(A(f(U_n), 1^n)\big)$ by definition of B, and the latter equality has probability $P(n)$ by definition of P.

So $\Pr[|B(1^n)| \geq n \wedge B(1^n)_1 \neq B(1^n)_2 \wedge f(B(1^n)_1) = f(B(1^n)_2)] = P(n)/2 \notin \mathcal{N}$ since $P(n) \notin \mathcal{N}$.

5.13. There is a mistake in the "proof" above, namely that the formula (∗) may be false (even assuming that (†) is true).

For example, if any collision (x_1, x_2) of f is such that $|x_1| \neq |x_2|$, then $(f(x), 1^{|x|})$ uniquely determines x (because x is the only preimage of $f(x)$ with length $|x|$), so $A(f(U_n), 1^n) = B(1^n)_2$, may determine $U_n = B(1^n)_1$ and output it, thus (‡) $A(f(U_n), 1^n) = B(1^n)_2 = U_n = B(1^n)_1$, hence $\Pr[B(1^n)_1 \neq B(1^n)_2] = 0$, therefore (∗) is false (while (†) is true).

The mistake in the "proof" of (†) $\Rightarrow$ (∗) is in the part "$B(1^n)_1$ has probability $1/2$ of being the other preimage because $B(1^n)_1$ is uniformly random". This quoted statement would be true if $B(1^n)_1$ and $B(1^n)_2$ were independent events, but they may be dependent events, they may even the same event as in (‡).

So the "proof" above is an example of "proof" that seems correct (we are confident that the reader did not notice the mistake when the reader read the "proof" — we also did not notice it when we first wrote the "proof") but turns out to be incorrect (having a subtle mistake that is easily overlooked). As such, the "proof" exemplifies our thesis that it is easy to make mistakes but hard to find them in proofs in cryptography.

5.14. A closer look at the mistake above shows that it depends crucially on the case "any collision (x_1, x_2) of f is such that $|x_1| \neq |x_2|$". So one way of correcting the proposition would be to add as a hypothesis the negation of the quoted case, or even better, its "strong negation" "any collision (x_1, x_2) of f is such that $|x_1| = |x_2|$". This motivates the following definition and corrected proposition.

5.15 Definition. We say that a binary-string function f is has *equal-length collisions* if and only if any collision (x_1, x_2) of f is such that $|x_1| = |x_2|$, that is $\forall x_1, x_2 \in \{0,1\}^*\ (f(x_1) = f(x_2) \ \Rightarrow\ |x_1| = |x_2|)$.

5.16 Proposition. For all binary-string functions f, if f is two-to-one, collision resistant and has equal-length collisions, then f is one-way.

5.17 Proof. The "proof" 5.12 is a correct proof under the assumption that f has equal-length collisions.

5.4 Example: $\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ versus $f \in \mathcal{N}$

5.18. Let us see an example showing how abbreviating a complex notion (such as $\forall \varepsilon > 0\ \exists N\ \forall n > N\ |x_n - l| < \varepsilon$) by a simple notation (such as $\lim x_n = l$) can simplify considerably a proof in cryptography. The example is the abbreviation of

$\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ by $f \in \mathcal{N}$. It comes in line with Dan Grundy and Eerke Boiten's introduction of a "for all large enough n" quantifier $\Diamond n\ P(n)$ abbreviating $\exists N\ \forall n > N\ P(n)$ (where N does not occur freely in $P(n)$) (Boiten and Grundy 2010, section 2).

Actually, there is more in the example than just an abbreviation: we use freely and without mention obvious properties of $\mathcal{N}$ such as $\forall f, g \in \mathcal{N}\ f + g \in \mathcal{N}$ and $\forall g \in \mathcal{N}\ \forall f \leq g\ f \in \mathcal{N}$, and obvious examples of elements of $\mathcal{N}$ such as $2^{-n} \in \mathcal{N}$ and $0 \in \mathcal{N}$. This mirrors what we do with probabilities: we use freely and without mention obvious properties of Pr such that $\forall A, B\ \Pr[A \vee B] = \Pr A + \Pr B - \Pr[A \wedge B]$ and $\forall B\ \forall A \subseteq B\ \ \Pr A \leq \Pr B$, and obvious examples of Pr such as $\Pr[U_n = 0^n] = 2^{-n}$ and $\Pr \emptyset = 0$.

Moreover, the search for complex notions to be abbreviated has the potential to lead to the discovery of new notions. For example if we did not already know the notion of negligibility, then our example could have led us to discover that notion.

In conclusion, the abbreviation of complex notions by simple notations can have the following three benefits:

1. to simplify the proof by replacing complex notions by simple abbreviations;

2. to further simply the proof by recasting verbose low-level arguments as more elegant high-level arguments;

3. to potentially lead to the discovery of new notions.

5.19. To compare the use of $\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ versus the use of $f \in \mathcal{N}$, we present the following proposition and two proofs of it, the first one using $\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n)$ and the second one using $f \in \mathcal{N}$. We trust that the reader will dislike the first proof because of its disadvantages such as alternating quantifiers $\forall\exists\forall$ and trivial but tiresome calculations pertaining to p and N, and will appreciate the second proof for abstracting away the said disadvantages behind the notation $f \in \mathcal{N}$. (Aside from providing us with an example to illustrate our thesis that the well-chosen notation considerably improves proof presentation, the proposition and its proofs are also of interest as a "warm-up" to the similar but slightly more complicated proposition 6.13 and its proof.)

5.20 Proposition. Let $O := \{0^n \mid n \in \mathbb{N}\}$. For all binary-string functions f, if f is one-way, then the binary-string function g defined by

$$g(x) := \begin{cases} \epsilon & \text{if } x \in O \\ f(x) & \text{if } x \notin O \end{cases}$$

is one-way.

5.21 Proof. The binary-string function f being one-way means that f is polynomial-time computable and

$$\forall A, p\ \exists M\ \forall n > M\ \Pr\big[f\big(A(f(U_n), 1^n)\big) = f(U_n)\big] < 1/p(n), \tag{5.2}$$

where A ranges over the polynomial-time probabilistic algorithms, p ranges over the positive polynomials, and M and n range over $\mathbb{N}$. The binary-string function g being one-way means that g is polynomial-time computable (which is the case because f is polynomial-time computable and O is polynomial-time decidable) and

$$\forall B, q\ \exists N\ \forall n > N\ \Pr\big[\underbrace{g\big(B(g(U_n), 1^n)\big) = g(U_n)}_{=:P}\big] < 1/q(n), \tag{5.3}$$

where B ranges over the polynomial-time probabilistic algorithms, q ranges over the positive polynomials, and N and n range over $\mathbb{N}$.

Let

$$\begin{aligned} Q &:= (U_n \in O), \\ R &:= \big(U_n \notin O\ \wedge\ B(g(U_n), 1^n) \in O\big), \\ S &:= \big(U_n \notin O\ \wedge\ B(g(U_n), 1^n) \notin O\big). \end{aligned}$$

To prove (5.3), we take arbitrary B and q, and we prove the following three claims, which later on we will see that imply (5.3).

<u>$\exists N'\ \forall n > N'\ \Pr[P\ \wedge\ Q] < 1/(3q(n))$</u> We have $\Pr Q = 2^{-n}$ because $|O \cap \{0,1\}^n| = 1$ and $|\{0,1\}^n| = 2^n$, so $\Pr[P\ \wedge\ Q] \leq 2^{-n}$. We have $\lim \big(1/(3q(n))\big)/2^{-n} = \lim 2^n/(3q(n)) = +\infty$ because an exponential eventually grows faster than a polynomial, so there is an N' such that $\forall n > N'\ \big(1/(3q(n))\big)/2^{-n} > 1$, that is $\forall n > N'\ 2^{-n} < 1/(3q(n))$, thus $\forall n > N'\ \Pr[P\ \wedge\ Q] \leq 2^{-n} < 1/(3q(n))$.

<u>$\exists N''\ \forall n > N''\ \Pr[P\ \wedge\ R] < 1/(3q(n))$</u> If $P\ \wedge\ R$, then $g(U_n) = f(U_n)$ and $g\big(B(g(U_n), 1^n)\big) = \epsilon$ by R, so $f(U_n) = \epsilon$ by P, thus $\Pr[P \wedge R] \leq \Pr[f(U_n) = \epsilon]$.

1. If $\nexists c \in \{0,1\}^*\ f(c) = \epsilon$, then $\forall n\ \Pr[f(U_n) = \epsilon] = 0 < 1/(3q(n))$.
2. If $\exists c \in \{0,1\}^*\ f(c) = \epsilon$, then taking $A(x, x') := c$, which is a probabilistic polynomial-time algorithm, and $p := 3q$, which is a positive polynomial because q is a positive polynomial, in (5.2), we get an N'' such that $\forall n > N''\ \Pr[f(U_n) = \epsilon] < 1/(3q(n))$.

So $\forall n > N''\ \Pr[f(U_n) = \epsilon] < 1/(3q(n))$ in any case, thus $\forall n > N''$ $\Pr[P\ \wedge\ R] \leq \Pr[f(U_n) = \epsilon] < 1/(3q(n))$.

<u>$\exists N'''\ \forall n > N'''\ \Pr[P\ \wedge\ S] < 1/(3q(n))$</u> If $P\ \wedge\ S$, then $g(U_n) = f(U_n)$ and $g\big(B(g(U_n), 1^n)\big) = f\big(B(f(U_n), 1^n)\big)$ by S, so $f\big(B(f(U_n), 1^n\big) = f(U_n)$ by P, thus $\Pr[P\ \wedge\ S] \leq \Pr\big[f\big(B(f(U_n), 1^n)\big) = f(U_n)\big]$. Taking $A := B$, which is a polynomial-time probabilistic algorithm because B is a polynomial-time probabilistic algorithm, and $p := 3q$, which is a positive polynomial because q is a positive polynomial, in (5.2), we get an N''' such that $\forall n > N'''$ $\Pr\big[f\big(B(f(U_n), 1^n)\big) = f(U_n)\big] < 1/(3q(n))$, so $\forall n > N'''\ \Pr[P\ \wedge\ S] \leq \Pr\big[f\big(B(f(U_n), 1^n)\big) = f(U_n)\big] < 1/(3q(n))$.

Finally, let us see that the three claims above imply (5.3): we have $P \Leftrightarrow (P\ \wedge\ Q) \vee (P\ \wedge\ R) \vee (P\ \wedge\ S)$, so taking $N := \max\{N', N'', N'''\}$, we get

$$\forall n > N\ \ \Pr P \leq \underbrace{\Pr[P\ \wedge\ Q]}_{<1/(3q(n))} + \underbrace{\Pr[P\ \wedge\ R]}_{<1/(3q(n))} + \underbrace{\Pr[P\ \wedge\ S]}_{<1/(3q(n))} < 1/q(n).$$

5.22 Proof. The binary-string function f being one-way means that f is polynomial-time computable and

$$\forall A \ \Pr\bigl[f\bigl(A(f(U_n), 1^n)\bigr) = f(U_n)\bigr] \in \mathcal{N}, \tag{5.4}$$

where A ranges over the polynomial-time probabilistic algorithms. The binary-string function g being one-way means that g is polynomial-time computable (which is the case because f is polynomial-time computable and O is polynomial-time decidable) and

$$\forall B \ \Pr\bigl[\underbrace{g\bigl(B(g(U_n), 1^n)\bigr) = g(U_n)}_{=:P}\bigr] \in \mathcal{N}, \tag{5.5}$$

where B ranges over the polynomial-time probabilistic algorithms.

Let

$$\begin{aligned} Q &:= (U_n \in O), \\ R &:= \bigl(U_n \notin O \ \wedge \ B(g(U_n), 1^n) \in O\bigr), \\ S &:= \bigl(U_n \notin O \ \wedge \ B(g(U_n), 1^n) \notin O\bigr). \end{aligned}$$

To prove (5.5), we notice $P \ \Leftrightarrow \ (P \ \wedge \ Q) \vee (P \ \wedge \ R) \vee (P \ \wedge \ S)$ and we prove $\Pr[P \ \wedge \ Q], \Pr[P \ \wedge \ R], \Pr[P \ \wedge \ S] \in \mathcal{N}$.

$\Pr[P \ \wedge \ Q] \in \mathcal{N}$ We have $\Pr Q = 2^{-n} \in \mathcal{N}$ because $|O \cap \{0,1\}^n| = 1$ and $|\{0,1\}^n| = 2^n$, so $\Pr[P \ \wedge \ Q] \in \mathcal{N}$.

$\Pr[P \ \wedge \ R] \in \mathcal{N}$ If $P \ \wedge \ R$, then $g(U_n) = f(U_n)$ and $g\bigl(B(g(U_n), 1^n)\bigr) = \epsilon$ by R, so $f(U_n) = \epsilon$ by P, thus $\Pr[P \ \wedge \ R] \leq \Pr[f(U_n) = \epsilon]$.

1. If $\nexists c \in \{0,1\}^* \ f(c) = \epsilon$, then $\Pr[f(U_n) = \epsilon] = 0 \in \mathcal{N}$.
2. If $\exists c \in \{0,1\}^* \ f(c) = \epsilon$, then taking $A(x, y) := c$, which is a probabilistic polynomial-time algorithm, in (5.4), we get $\Pr[f(U_n) = \epsilon] \in \mathcal{N}$.

So $\Pr[f(U_n) = \epsilon] \in \mathcal{N}$ in any case, thus $\Pr[P \ \wedge \ R] \in \mathcal{N}$.

$\Pr[P \ \wedge \ S] \in \mathcal{N}$ If $P \wedge S$, then $g(U_n) = f(U_n)$ and $g\bigl(B(g(U_n), 1^n)\bigr) = f\bigl(B(f(U_n), 1^n)\bigr)$ by S, so $f\bigl(B(f(U_n), 1^n\bigr) = f(U_n)$ by P, thus $\Pr[P \ \wedge \ S] \leq \Pr\bigl[f\bigl(B(f(U_n), 1^n)\bigr) = f(U_n)\bigr]$. Taking $A := B$, which is a polynomial-time probabilistic algorithm because B is a polynomial-time probabilistic algorithm, in (5.4), we get $\Pr\bigl[f\bigl(B(f(U_n), 1^n)\bigr) = f(U_n)\bigr] \in \mathcal{N}$, so $\Pr[P \ \wedge \ S] \in \mathcal{N}$.

5.5 Conclusion

5.23. In this chapter we presented three examples of proof presentation:

1. a bad definition;
2. an incorrect cryptographic proof;
3. an improvement of a proof.

Part IV

Transformations and proof presentations

Chapter 6

Transforming one-way length-nondecreasing binary-string functions into (possibly different) one-way binary-string functions

6.1 Introduction

6.1. In this chapter we solve the instance of our problem format (presented in paragraph 1.28) specified by the following instances of its parameters:

$$\begin{aligned}
P_1 &= \text{length-nondecreasing binary-string function,}\\
S_1 &= \text{one-wayness,}\\
P_2 &= \text{binary-string function,}\\
S_2 &= \text{one-wayness,}\\
T &= \text{post-composition with a fixed}\\
&\quad\ \text{collision-resistant binary-string function.}
\end{aligned}$$

As a proof presentation, we make the security proof of T more understandable (but less formal) by pulling to the foreground the proof idea and pushing to the background the technical details.

As an extra, we give a counterexample to the statement "the composition of two one-way binary-string functions is one-way" (assuming the existence of one-way functions).

6.2. Let us informally explain our instances of the parameters.

Length-nondecreasing binary-string function A binary-string function is a function that inputs and outputs binary strings. Being length nondecreasing means that each output is not shorter than the corresponding input.

One-wayness It means that the function f is easy to compute but hard to invert, that is $x \mapsto f(x)$ is easy to compute but $f(x) \mapsto x$ (or more precisely, $f(x) \mapsto x'$ with $f(x) = f(x'))$ is hard to compute.

Transformation A collision-resistant function f is such that it is difficult to find collisions of f, that is pairs (x, x') of distinct inputs with the same output, or in other words, such that $x \neq x'$ and $f(x) = f(x')$. Post-composition means that f appears on the left side of the symbol $\circ$ in $f \circ g$. The transformation inputs a length-nondecreasing binary-string function g and and outputs the binary-string function $f \circ g$.

6.3. In this chapter:

1. almost all content is heavily based on joint work with Eerke Boiten (Gaspar and Boiten 2014);

2. the counterexample is based on a statement by Pooya Farshim, which was strengthened and proved by us.

6.2 Transformation

6.4. Now, to be sure, we explicitly state the transformation in question.

6.5 Definition. Let f be a fixed collision-resistant binary-string function. The *transformation* of a length-nondecreasing binary-string function g into a (possibly different) binary-string function $f \circ g$ by post-composition with the collision-resistant binary-string function f is $g \rightsquigarrow f \circ g$.

6.3 Security

6.6. Now we show that if a length-nondecreasing binary-string function g is one-way, then the binary-string function $f \circ g$ obtained by post-composition with a collision-resistant binary-string function f is one-way. Informally, this means that secure (in some sense) length-nondecreasing binary-string functions are transformed into secure (also in some sense) binary-string functions.

6.7 Theorem. Let f be a fixed collision-resistant binary-string function. For all length-nondecreasing binary-string functions g, if g is one-way, then $f \circ g$ is one-way.

6.8 Proof. The binary-string function f being collision resistant means that f is polynomial-time computable and

$$\forall A \ \Pr[|A(1^n)| \geq n \wedge A(1^n)_1 \neq A(1^n)_2 \wedge f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N}, \tag{6.1}$$

where A ranges over the polynomial-time probabilistic algorithms. The length-nondecreasing binary-string function g being one-way means that g is polynomial-time computable and

$$\forall B \ \Pr\big[g\big(B(g(U_n), 1^n)\big) = g(U_n)\big] \in \mathcal{N}, \tag{6.2}$$

where B ranges over the polynomial-time probabilistic algorithms. The binary-string function $f \circ g$ being one-way means that $f \circ g$ is polynomial-time computable, which is the case because f and g are polynomial-time computable, and

$$\forall C \ \Pr\big[\underbrace{f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)}_{\Leftrightarrow :P}\big] \in \mathcal{N}, \tag{6.3}$$

where C ranges over the polynomial-time probabilistic algorithms.

Taking $A(x) := \big(g\big(C(f \circ g(U_{|x|}), x)\big), g(U_{|x|})\big)$, which is a polynomial-time probabilistic algorithm because C and $U.$ are polynomial-time probabilistic algorithm and f, g, $|\cdot|$ and $(\cdot, \cdot)$ are polynomial-time computable, in (6.1), and $B(x, y) := C(f(x), y)$, which is a polynomial-time probabilistic algorithm because C is a polynomial-time probabilistic algorithm and f is polynomial-time computable, in (6.2), we get

$$\forall C \ \Pr\big[\overbrace{\big|g\big(C(f \circ g(U_n), 1^n)\big)\big| + |g(U_n)| \geq n}^{(*)} \wedge$$
$$g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \ f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big] \in \mathcal{N},$$
$$\forall C \ \Pr\big[g\big(C(f \circ g(U_n), 1^n)\big) = g(U_n)\big] \in \mathcal{N}.$$

We have $(*)$ because g is length nondecreasing, so

$$\forall C \ \Pr\big[\overbrace{g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \ f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)}^{\Leftrightarrow :Q}\big] \in \mathcal{N},$$
$$\forall C \ \Pr\big[\underbrace{g\big(C(f \circ g(U_n), 1^n)\big) = g(U_n)}_{\Leftrightarrow :R}\big] \in \mathcal{N}.$$

Using $\Pr[Q \vee R] \in \mathcal{N}$ because $\Pr[Q \vee R] \leq \Pr Q + \Pr R \in \mathcal{N}$ since $\Pr Q, \Pr R \in \mathcal{N}$, we get

$$\forall C \ \Pr\big[\big(\overbrace{g\big(C(f \circ g(U_n), 1^n)\big)}^{=:x} \neq \overbrace{g(U_n)}^{=:x'} \wedge f \circ \overbrace{g\big(C(f \circ g(U_n), 1^n)\big)}^{=x} = \underline{f \circ \overbrace{g(U_n)}^{=x'}}\big) \ \vee$$
$$\underbrace{g\big(C(f \circ g(U_n), 1^n)\big)}_{=x} = \underbrace{g(U_n)}_{=x'}\big] \in \mathcal{N},$$

where the underlined part means $f(x')$ and not $f \circ x'$, and analogously elsewhere. Using $(x \neq x' \ \wedge \ f(x) = f(x')) \ \vee \ x = x' \ \Leftrightarrow \ f(x) = f(x')$, we get (6.3).

6.4 Proof presentation: proof idea

6.9. We will improve the proof presentation of proof 6.8 (motivated by the reasons given in section 1.5).

6.10 Proof. Let

$$\begin{aligned} P \ &:\Leftrightarrow \ f \circ \overbrace{g\big(C(f \circ g(U_n), 1^n)\big)}^{=:x} = f \circ \overbrace{g(U_n)}^{=:x'}, \\ Q \ &:\Leftrightarrow \ x = x' \ \wedge \ P, \\ R \ &:\Leftrightarrow \ x \neq x' \ \wedge \ P. \end{aligned}$$

To prove that $f \circ g$ is one-way we have essentially to prove $\Pr P \in \mathcal{N}$ (for all polynomial-time probabilistic algorithms C). To do this, we notice $P \Leftrightarrow Q \vee R$ and we prove $\Pr Q, \Pr R \in \mathcal{N}$.

$\Pr Q \in \mathcal{N}$ If Q, then $x = x'$, so $B(x, y) := C(f(x), y)$ is inverting g by P, which by the one-wayness of g has negligible probability.

$\Pr R \in \mathcal{N}$ If R, then

1. the ordered pair (x, x') is a collision of f by P;
2. the collision (x, x') is arbitrarily long because $|x'| \geq n$ since g is length nondecreasing;

which by the collision resistance of f has negligible probability.

6.11. We see two advantages in proof 6.10 over proof 6.8:

1. it shows that the core of the argument is that
   (a) $P \Leftrightarrow Q \vee R$;
   (b) $\Pr Q \in \mathcal{N}$ (because Q implies that B is inverting the one-way function g);
   (c) $\Pr R \in \mathcal{N}$ (because R essentially implies that (x, x') is a collision of the collision-resistant function f);
2. it avoids the complicated formulas and manipulations from proof 6.8.

6.5 Extra: composition of one-way functions is not necessarily one-way

6.12. During an investigation of "algebras" consisting of one-way functions and other functions under the composition operation, we considered the following question: when is the composition of a one-way function f or g with another function g or f a one-way function $f \circ g$?

1. Eerke Boiten and Dan Grundy gave the following sufficient condition for pre-composition: if
   (a) f is a one-way binary-string function;
   (b) g is a polynomial-to-one length-preserving binary-string function;

   then the binary-string function $f \circ g$ is one-way (Grundy and Boiten 2008, paragraph "One-way functions and pre-composition").
2. We gave the following sufficient condition for post-composition: if
   (a) f is a collision-resistant binary-string function;
   (b) g is a one-way length-nondecreasing binary-string function;

then the binary-string function $f \circ g$ is one-way (theorem 6.7).

3. But what about if both f and g are one-way?

We address this question in the following proposition by giving a negative answer (if one-way functions exists), even in the case $f = g$, using a counterexample essentially by Pooya Farshim (we simplified and adapted the counterexample) and we give a detailed proof of it by us (we do not know if Pooya Farshim had a proof in mind or relied on intuition). Informally, the counterexample consists in considering a one-way function f, a "negligible" set $N \subseteq \{0,1\}^*$, and constructing another one-way functions g such that the image of g is contained in N and g restricted to N is a constant function, so $g \circ g$ is a constant function and therefore is not one-way. The construction uses the "double" function d that doubles the length of an input x by prepending $0^{|x|}$ to x. (The counterexample and its proof are similar but slightly more complicated than proposition 5.20 its proof 5.22.)

6.13 Proposition. Let d be the binary-string function defined by $d(x) := 0^{|x|}x$ and $N := \operatorname{im}(d)$. For all binary-string functions f, if f is one-way, then the binary-string function g defined by

$$g(x) := \begin{cases} \epsilon & \text{if } x \in N \\ d(f(x)) & \text{if } x \notin N \end{cases}$$

is one-way but the binary-string function $g \circ g$ is not one-way (Farshim 2015, based on his counterexample).

6.14 Proof. It is easy to prove that $g \circ g$ is not one-way: we have $\forall x \in \{0,1\}^*$ $g(x) \in N$ and $\forall x \in N$ $g(x) = \epsilon$, so $\forall x \in \{0,1\}^*$ $g \circ g(x) = \epsilon$, thus $g \circ g$ is invertible with probability 1 by any polynomial-time probabilistic algorithm. So let us focus on proving that g is one-way (assuming that f is one-way).

The binary-string function f being one-way means that f is polynomial-time computable and

$$\forall A \ \Pr\big[f\big(A(f(U_n), 1^n)\big) = f(U_n)\big] \in \mathcal{N}, \tag{6.4}$$

where A ranges over the polynomial-time probabilistic algorithms. The binary-string function g being one-way means that g is polynomial-time computable, which is the case because d and f are polynomial-time computable and N is polynomial-time decidable, and

$$\forall B \ \Pr\big[\underbrace{g\big(B(g(U_n), 1^n)\big) = g(U_n)}_{\Leftrightarrow: P}\big] \in \mathcal{N}, \tag{6.5}$$

where B ranges over the polynomial-time probabilistic algorithms.

Let

$$\begin{aligned} Q \ &:\Leftrightarrow \ U_n \in N, \\ R \ &:\Leftrightarrow \ U_n \notin N \ \wedge \ B(g(U_n), 1^n) \in N, \\ S \ &:\Leftrightarrow \ U_n \notin N \ \wedge \ B(g(U_n), 1^n) \notin N. \end{aligned}$$

To prove (6.5), we notice $P \ \Leftrightarrow \ (P \wedge Q) \vee (P \wedge R) \vee (P \wedge S)$ and we prove $\Pr[P \wedge Q], \Pr[P \wedge R], \Pr[P \wedge S] \in \mathcal{N}$.

$\underline{\Pr[P \wedge Q] \in \mathcal{N}}$

1. If n is odd, then $\Pr Q = 0 \in \mathcal{N}$ because all elements of N have even length.
2. If n is even, then $\Pr Q = 2^{-n/2} \in \mathcal{N}$ because $|N \cap \{0,1\}^n| = 2^{n/2}$ and $|\{0,1\}^n| = 2^n$.

So $\Pr Q \in \mathcal{N}$, thus $\Pr[P \wedge Q] \in \mathcal{N}$.

$\underline{\Pr[P \wedge R] \in \mathcal{N}}$ If $P \wedge R$, then $g(U_n) = d(f(U_n))$ and $g\big(B(g(U_n), 1^n)\big) = \epsilon$ by R, so $f(U_n) = \epsilon$ by P, thus $\Pr[P \wedge R] \leq \Pr[f(U_n) = \epsilon]$.

1. If $\nexists c \in \{0,1\}^*\ f(c) = \epsilon$, then $\Pr[f(U_n) = \epsilon] = 0 \in \mathcal{N}$.
2. If $\exists c \in \{0,1\}^*\ f(c) = \epsilon$, then taking $A(x,y) := c$, which is a probabilistic polynomial-time algorithm, in (6.4), we get $\Pr[f(U_n) = \epsilon] \in \mathcal{N}$.

So $\Pr[f(U_n) = \epsilon] \in \mathcal{N}$, thus $\Pr[P \wedge R] \in \mathcal{N}$.

$\underline{\Pr[P \wedge S] \in \mathcal{N}}$ If $P \wedge S$, then $g(U_n) = d(f(U_n))$ and $g\big(B(g(U_n), 1^n)\big) = d\Big(f\big(B\big(d(f(U_n)), 1^n\big)\big)\Big)$ by S, so $f\big(B\big(d(f(U_n)), 1^n\big)\big) = f(U_n)$ by P, thus $\Pr[P \wedge S] \leq \Pr\big[f\big(B\big(d(f(U_n)), 1^n\big)\big) = f(U_n)\big]$. Taking $A(x,y) := B(d(x), y)$, which is a polynomial-time probabilistic algorithm because B is a polynomial-time probabilistic algorithm and d is polynomial-time computable, in (6.4), we get $\Pr\big[f\big(B\big(d(f(U_n)), 1^n\big)\big) = f(U_n)\big] \in \mathcal{N}$, so $\Pr[P \wedge S] \in \mathcal{N}$.

6.6 Conclusion

6.15. In this chapter:

1. we solved the instance of our problem specified by

$$\begin{aligned} P_1 &= \text{length-nondecreasing binary-string function,} \\ S_1 &= \text{one-wayness,} \\ P_2 &= \text{binary-string function,} \\ S_2 &= \text{one-wayness,} \\ T &= \text{post-composition with a fixed} \\ &\quad\ \text{collision-resistant binary-string function;} \end{aligned}$$

2. we gave a proof presentation by pulling to the foreground the proof idea and pushing to the background the technical details;

3. we gave a counterexample to the statement "the composition of two one-way binary-string functions is one-way".

Chapter 7

Transforming cryptographically-secure pseudorandom generators into indistinguishable-from-random stream ciphers

7.1 Introduction

7.1. In this chapter we solve the instance of our problem format (presented in paragraph 1.28) specified by the following instances of its parameters (Gaspar 2016, section 1, taken almost verbatim):

$$\begin{aligned} P_1 &= \text{pseudorandom generator,} \\ S_1 &= \text{cryptographic security,} \\ P_2 &= \text{stream cipher,} \\ S_2 &= \text{indistinguishability from random,} \\ T &= \text{transformation of a pseudorandom generator} \\ &\quad\ \text{into its induced stream cipher.} \end{aligned}$$

As proof presentations, we

1. give a security proof of T using a schematic proof;
2. give another security proof of T using the wedding-cake notation;
3. "compress" nine analogous and verbose claims into a shorter presentation.

As extras, we prove that indistinguishability from random implies

1. cryptographic security;
2. indistinguishable encryptions;

3. semantic security;
4. bit-recovery resistance;

and we comment on

1. the reciprocal implications;
2. the role of length regularity of a stream cipher;

for the last three implications.

7.2. Let us informally explain our instances of the parameters.

Pseudorandom generator It is a deterministic algorithm that outputs a stream of bits such as 001101100.

Cryptographic security It means that the stream output by the pseudorandom generator looks random such as 0011011001 in contrast to 0101010101.

Stream cipher It is a cipher that mixes a plaintext such as 0000011111 with the stream output by a pseudorandom generator such as 0011011001 to create a ciphertext such as 0011000110.

Indistinguishability from random It means that for all plaintexts such as 0000011111 chosen by an adversary, the corresponding ciphertexts computed by the stream cipher look random such as 0011000110 in contrast to 0101010101.

Transformation It inputs a pseudorandom generator and outputs the stream cipher that creates a ciphertext by mixing a plaintext with the stream output by the pseudorandom generator.

7.3. In this chapter:

1. all theorems and proofs are ours;
2. the content of sections 7.2, 7.3 and 7.6 is based on an informal publication of ours (Gaspar 2016).

7.2 Transformation

7.4. Let us recall that a stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ is length regular if and only if it encrypts plaintexts of equal length into ciphertexts of equal length even under different keys, that is

$$\forall k, k' \in \mathcal{K}\ \forall p, p' \in \mathcal{P}\ (|p| = |p'| \ \Rightarrow\ |E(k, p)| = |E(k', p')|).$$

We can think of length regularity as a modest security notion saying that the lengths of the ciphertexts alone do not reveal a difference between the plaintexts (but an analysis of the content of the plaintexts may reveal something). Or to rephrase things negatively and giving an example, if a stream cipher were not length regular and would encrypt $p := 0$ as $c := E(k,p) = 0$ and $p' := 1$ as $c' := E(k,p') = 00$ (notice $|p| = |p'|$ but $|c| \neq |c'|$), and we were given one of the ciphertexts c and c', then just looking at the length of the ciphertext we could deduce whether the corresponding plaintext is p or p'.

Length regularity is a technical condition appearing often in definitions, theorems and propositions below. Since it is not of much interest on its own (because it is a modest security notion), we will treat it as a "second-class citizen" by mostly remitting it to remarks after the definitions, theorems and propositions.

7.5. Let us introduce, by an example, a construction of a stream cipher induced by a pseudorandom generator (a cipher which we could also call pseudo one-time pad because it is another cipher called one-time pad but with a random key replaced by a pseudorandom key).

For example if our plaintext is 0000011111 and our stream of the pseudorandom generator is 0011011001, then we can encrypt and get the ciphertext 0011000110 schematically calculated by xoring as

$$\begin{array}{rcccccccccccl}
\text{plaintext} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \\
 & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \\
\text{stream} & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \\
 & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \\
 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \text{ciphertext}
\end{array}$$

and from the ciphertext 0011000110 and the same stream 0011011001 we can decrypt and recover our plaintext 0000011111 schematically by xoring again as

$$\begin{array}{rcccccccccccl}
\text{ciphertext} & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \\
 & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \\
\text{stream} & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \\
 & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \parallel & \\
 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \text{plaintext}
\end{array}$$

Let us observe that the stream acts as a key to encrypt and decrypt. Also let us notice that the stream needs to be as long as the plaintext. So if the plaintext is very long, then it may be too onerous to store the entire stream (this is a well-known practical limitation of the one-time pad). Thus it is more practical to store the seed used by the pseudorandom generator to produce the stream as the key and to recreate the stream from the seed when necessary.

7.6. Informally, a stream cipher induced by a pseudorandom generator G is the cipher that inputs a key k and a plaintext p, passes k as seed to G to get a stream $g := G(k, 1^{|p|})$ with the same length as p, encrypts p by xoring it g giving the ciphertext $c := g \oplus p$, and decrypts c by xoring again with g giving the plaintext $c \oplus g = p \oplus g \oplus g = p$.

Now we formally define the stream cipher induced by a pseudorandom generator.

7.7 Definition. The *stream cipher* $C_G = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E_G, D_G)$ *induced by the pseudorandom generator* G (Katz and Lindell 2015, construction 3.17) is the stream cipher defined by

1. $\forall x \in \{0,1\}^* \; K(x) := U_{|x|}$;
2. $\forall k \in \mathcal{K} \; \forall p \in \mathcal{P} \; E_G(k,p) := G(k, 1^{|p|}) \oplus p$;
3. $D_G := E_G$.

7.8 Remark. The stream cipher C_G induced by the pseudorandom generator G is length regular.

7.9. Now, to be sure, we explicitly state the transformation in question.

7.10 Definition. The *transformation* of a pseudorandom generator G into its induced stream cipher C_G is $G \rightsquigarrow C_G$.

7.3 Security

7.11. Now we show that if the pseudorandom generator G is cryptographically secure, then its induced stream cipher C_G is indistinguishable from random. Informally, this means that secure (in some sense) pseudorandom generators are transformed into secure (also in some sense) stream ciphers.

7.12 Theorem. For all pseudorandom generators G, if G is cryptographically secure, then C_G is indistinguishable from random.

7.13 Proof. The pseudorandom generator G being cryptographically secure means that

$$\begin{gathered}\forall A, A' \; \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \\ \Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N},\end{gathered} \tag{7.1}$$

where A and A' range over the polynomial-time probabilistic algorithms. The stream cipher C_G being indistinguishable from random means that

$$\begin{gathered}\forall B, B' \; \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\ \Pr B'(U'_{|E_G(K(1^n), B(1^n)_1)|}, 1^n, B(1^n)_2) \in \mathcal{N},\end{gathered} \tag{7.2}$$

where B and B' range over the polynomial-time probabilistic algorithms.

Taking $A(x) := (B(x)_1, B(x))$ and $A'(x,y,z) := B'(x \oplus z_1, y, z_2)$, which are polynomial-time probabilistic algorithms because B and B' are polynomial-time probabilistic algorithms and $\cdot_1$, $\cdot_2$, $(\cdot,\cdot)$ and $\oplus$ are polynomial-time computable, in (7.1), we get

$$\begin{gathered}\forall B, B' \; \Pr B'(G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1, 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n, 1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{gathered}$$

Substituting $G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1$ by $E_G(U_n, B(1^n)_1)$, by definition of E_G, we get

$$\begin{gathered}\forall B, B' \; \Pr B'\big(E_G(U_n, B(1^n)_1), 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|G(U_n, 1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{gathered}$$

Substituting U_n by $K(1^n)$, by definition of K, we get

$$\forall B, B' \ \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|G(K(1^n),1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Substituting $U_{|G(K(1^n),1^{|B(1^n)_1|})|} \oplus B(1^n)_1$, which is a uniform random variable in $\{0,1\}^{|G(K(1^n),1^{|B(1^n)_1|})|}$ because $U_{|G(K(1^n),1^{|B(1^n)_1|})|}$ and $B(1^n)_1$ are independent since the former only uses the length of the latter, by $U'_{|G(K(1^n),1^{|B(1^n)_1|})|}$, we get

$$\forall B, B' \ \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\ \Pr B'(U'_{|G(K(1^n),1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Substituting $|G(K(1^n), 1^{|B(1^n)_1|})|$ by $|E_G(K(1^n), B(1^n)_1)|$, by definition of G and E_G, we get (7.2).

7.14. It is worth remarking that the part "Substituting $U_{|G(K(1^n),1^{|B(1^n)_1|})|} \oplus B(1^n)_1$, which is a uniform random variable in $\{0,1\}^{|G(K(1^n),1^{|B(1^n)_1|})|}$ because $U_{|G(K(1^n),1^{|B(1^n)_1|})|}$ and $B(1^n)_1$ are independent [...], by $U'_{|G(K(1^n),1^{|B(1^n)_1|})|}$" of proof 7.13 uses a recurrent fact in cryptography: a uniform random variable U_n in $\{0,1\}^n$ xored with an *independent* random variable X_n in $\{0,1\}^n$ gives a uniform random variable U'_n in $\{0,1\}^n$, or less precisely but more succinctly, $U_n \perp X_n \ \Rightarrow \ U_n \oplus X_n = U'_n$.

7.4 Proof presentation: schematic proof

7.15. We are going to improve the proof presentation of proof 7.13 (motivated by the reasons given in section 1.5).

7.16 Proof.

$$G \text{ is cryptographically secure}$$

$$\Big\Updownarrow \ \text{Definition}$$

$$\forall A, A' \ \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \Pr A'(U_{|G(U_n,1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N}$$

$$\Big\Downarrow \ \begin{array}{c} A(x) := (B(x)_1, B(x)) \\ A'(x,y,z) := B'(x \oplus z_1, y, z_2) \end{array}$$

$$\forall B, B' \ \Pr B'(G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1, 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}$$

$$\Big\Updownarrow \ \begin{array}{c} G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1 = \\ E_G(U_n, B(1^n)_1) \end{array}$$

$$\forall B, B' \ \Pr B'\big(E_G(U_n, B(1^n)_1), 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}$$

$$\Big\Updownarrow \ U_n = K(1^n)$$

$$\begin{gathered}
\forall B, B' \;\; \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\
\Pr B'(U_{|G(K(1^n), 1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N} \\
\Big\Updownarrow \; \begin{array}{c} U_{|G(K(1^n), 1^{|B(1^n)_1|})|} \oplus B(1^n)_1 = \\ U'_{|G(K(1^n), 1^{|B(1^n)_1|})|} \end{array} \\
\forall B, B' \;\; \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\
\Pr B'(U'_{|G(K(1^n), 1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) \in \mathcal{N} \\
\Big\Updownarrow \; \begin{array}{c} |G(K(1^n), 1^{|B(1^n)_1|})| = \\ |E_G(K(1^n), B(1^n)_1)| \end{array} \\
\forall B, B' \;\; \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\
\Pr B'(U'_{|E_G(K(1^n), B(1^n)_1)|}, 1^n, B(1^n)_2) \in \mathcal{N} \\
\Big\Updownarrow \; \text{Definition} \\
C_G \text{ is indistinguishable from random}
\end{gathered}$$

7.17. In proof 7.16 we indicated the second step as being an implication instead of an equivalence. This is because that step consists in instantiations for which it is not obvious how to revert them. But in fact they can be reverted and this is done below in proof 7.26.

7.18. We see two advantages in proof 7.16 over proof 7.13:

1. it shows clearly the linear structure of the proof, in which a formula expressing the premise "G is cryptographically secure" is "massaged" step by step until we obtain a formula expressing the conclusion "C_G is indistinguishable from random";

2. it is more appealing because it is presented (partially) as a diagram instead of (fully) as text.

7.19. We argued that the schematic proof 7.16 is better than the conventional proof 7.13 but we will keep using the conventional proof because it includes full justifications, which are missing in the schematic proof due to the graphical restrictions of its schematic nature.

7.5 Proof presentation: wedding-cake notation

7.20. We are going to improve the proof presentation of proof 7.13 (motivated by the reasons given in section 1.5).

7.21 Proof. Let us start with the following formula expressing that the pseudorandom generator G is cryptographically secure:

$$\forall A, A' \;\; \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N}.$$

We are going to substitute in it some subformulas by other subformulas using the following equalities labelled from α to ε:

$$\begin{aligned}
\alpha&\colon\ A(x) := (B(x)_1, B(x)),\ \ A'(x,y,z) := B'(x \oplus z_1, y, z_2),\\
\beta&\colon\ G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1 = E_G(U_n, B(1^n)_1),\\
\gamma&\colon\ U_n = K(1^n),\\
\delta&\colon\ U_{|G(K(1^n),1^{|B(1^n)_1|})|} \oplus B(1^n)_1 = U'_{|G(K(1^n),1^{|B(1^n)_1|})|},\\
\varepsilon&\colon\ |G(K(1^n), 1^{|B(1^n)_1|})| = |E_G(K(1^n), B(1^n)_1)|.
\end{aligned}$$

We do the substitutions schematically using the notation

$$\underbrace{SF}_{\zeta\; SF'}$$

to indicate that subformula SF was replaced by subformula SF' using equality ζ:

$$\forall \underbrace{A, A'}_{\alpha\; B, B'}\ \Pr \underbrace{A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2)}_{\alpha\; B'(\underbrace{G(U_n,1^{|B(1^n)_1|}) \oplus B(1^n)_1}_{\beta\; E_G(\underbrace{U_n}_{\gamma\; K(1^n)}, B(1^n)_1)}, 1^n, B(1^n)_2)} - \Pr \underbrace{A'(U_{|G(U_n,1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2)}_{\alpha\; B'(\underbrace{U_{|G(\underbrace{U_n}_{\gamma\; K(1^n)},1^{|B(1^n)_1|})|} \oplus B(1^n)_1}_{\delta\; U'_{\underbrace{|G(K(1^n),1^{|B(1^n)_1|})|}_{\varepsilon\; |E_G(K(1^n),B(1^n)_1)|}}}, 1^n, B(1^n)_2)} \in \mathcal{N}$$

After all substitutions, we get the underdotted formula above spelling

$$\begin{gathered}
\forall B, B'\ \ \Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big) - \\
\Pr B'(U'_{|E_G(K(1^n),B(1^n)_1)|}, 1^n, B(1^n)_2) \in \mathcal{N},
\end{gathered}$$

which expresses that the stream cipher C_G is indistinguishable from random.

7.22. We see two advantages in proof 7.21 over proof 7.13 and even over proof 7.16:

1. it avoids "dragging along" a long formula that is essentially repeated from instance to instance with only minor changes, resulting in a shortening from approximately one page to approximately half a page (consisting of the equalities labelled from α to ε plus the wedding-cake diagram);

2. it is more appealing because it is presented (partially) as a diagram instead of (fully) as text.

7.23. What was said in paragraph 7.19 still applies here.

7.6 Extra: reciprocal implication

7.24. In theorem 7.12 we saw that if a pseudorandom generator G is cryptographically secure, then the stream cipher C_G is indistinguishable from random. Now let us see the reciprocal implication.

7.25 Theorem. For all pseudorandom generators G, if C_G is indistinguishable from random, then G is cryptographically secure.

7.26 Proof. The stream cipher C_G being indistinguishable from random means that

$$\begin{aligned}\forall A, A' \;\; \Pr A'\big(E_G(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\big) - \\ \Pr A'(U_{|E_G(K(1^n),A(1^n)_1)|}, 1^n, A(1^n)_2) \in \mathcal{N},\end{aligned} \tag{7.3}$$

where A and A' range over the polynomial-time probabilistic algorithms. The pseudorandom generator G being cryptographically secure means that

$$\begin{aligned}\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}), 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) \in \mathcal{N},\end{aligned} \tag{7.4}$$

where B and B' range over the polynomial-time probabilistic algorithms.

Taking $A(x) := (0^{|B(x)_1|}, B(x)_2)$ and $A' := B'$, which are polynomial-time probabilistic algorithms because B and B' are polynomial-time deterministic algorithm and $\cdot_1$, $\cdot_2$, $(\cdot,\cdot)$, $|\cdot|$ and $0^{\cdot}$ are polynomial-time computable, in (7.3), we get

$$\begin{aligned}\forall B, B' \;\; \Pr B'\big(E_G(K(1^n), 0^{|B(1^n)_1|}), 1^n, B(1^n)_2\big) \\ \Pr B'(U_{|E_G(K(1^n),0^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) \in \mathcal{N}.\end{aligned}$$

Substituting $E_G(K(1^n), 0^{|B(1^n)_1|})$ by $G(K(1^n), 1^{|B(1^n)_1|}) \oplus 0^{|B(1^n)_1|}$, by definition of E_G, we get

$$\begin{aligned}\forall B, B' \;\; \Pr B'\big(G(K(1^n), 1^{|B(1^n)_1|}) \oplus 0^{|B(1^n)_1|}, 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|G(K(1^n),1^{|B(1^n)_1|})\oplus 0^{|B(1^n)_1|}|}, 1^n, B(1^n)_2) \in \mathcal{N}.\end{aligned}$$

Substituting $K(1^n)$ by U_n, by definition of K, we get

$$\begin{aligned}\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}) \oplus 0^{|B(1^n)_1|}, 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})\oplus 0^{|B(1^n)_1|}|}, 1^n, B(1^n)_2) \in \mathcal{N}.\end{aligned}$$

Substituting $G(U_n, 1^{|B(1^n)_1|}) \oplus 0^{|B(1^n)_1|}$ by $G(U_n, 1^{|B(1^n)_1|})$, by definition of $\oplus$, we get (7.4).

Alternatively, taking $A(x) := (B(x)_1, B(x))$ and $A'(x, y, z) := B(x \oplus z_1, y, z_2)$, which are polynomial-time probabilistic algorithms because A and B are polynomial-time probabilistic algorithms and $\cdot_1$, $\cdot_2$, $(\cdot,\cdot)$ and $\oplus$ are polynomial-time computable, in (7.3), we get

$$\begin{aligned}\forall B, B' \;\; \Pr B'\big(E_G(K(1^n), B(1^n)_1) \oplus B(1^n)_1, 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|E_G(K(1^n),B(1^n)_1)|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{aligned}$$

Substituting $K(1^n)$ by U_n, by definition of K, we get

$$\begin{gathered}\forall B, B' \;\; \Pr B'\big(E_G(U_n, B(1^n)_1) \oplus B(1^n)_1, 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|E_G(U_n,B(1^n)_1)B(1^n)_1|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{gathered}$$

Substituting $E_G(U_n, B(1^n)_1)$ by $G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1$, by definition of E_G, we get

$$\begin{gathered}\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1 \oplus B(1^n)_1, 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})\oplus B(1^n)_1|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{gathered}$$

Substituting $G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1 \oplus B(1^n)_1$ by $G(U_n, 1^{|B(1^n)_1|})$, by definition of $\oplus$, we get

$$\begin{gathered}\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}), 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})\oplus B(1^n)_1|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{gathered}$$

Substituting $|G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1|$ by $|G(U_n, 1^{|B(1^n)_1|})|$, by definition of $\oplus$, we get

$$\begin{gathered}\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}), 1^n, B(1^n)_2) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.\end{gathered}$$

Substituting $U_{|G(U_n,1^{|B(1^n)_1|})|} \oplus B(1^n)_1$, which is a uniform random variable in $\{0,1\}^{|G(U_n,1^{|B(1^n)_1|})|}$ because $U_{|G(U_n,1^{|B(1^n)_1|})|}$ and $B(1^n)_1$ are independent since the former only uses the length of the latter, by $U'_{|G(U_n,1^{|B(1^n)_1|})|}$, we get

$$\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}), 1^n, B(1^n)_2) - \Pr B'(U'_{|G(U_n,1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Substituting $U'_{|G(K(1^n),1^{|B(1^n)_1|})|}$ by $U_{|G(K(1^n),1^{|B(1^n)_1|})|}$, because they both denote uniform random variables in $\{0,1\}^{|G(K(1^n),1^{|B(1^n)_1|})|}$ and there is no occurrence of the latter variable, we get (7.4).

7.7 Extra: indistinguishable encryptions

7.27. Let us prove that indistinguishability from random implies indistinguishable encryptions.

7.28 Proposition. For all length-regular stream ciphers C, if C is indistinguishable from random, then C has indistinguishable encryptions.

7.29 Proof. Say $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$. The stream cipher C being indistinguishable from random means that

$$\begin{gathered}\forall A, A' \;\; \Pr A'\big(E(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\big) - \\ \Pr A'(U_{|E(K(1^n),A(1^n)_1)|}, 1^n, A(1^n)_2) \in \mathcal{N},\end{gathered} \tag{7.5}$$

where A and A' range over the polynomial-time probabilistic algorithms. The stream cipher C having indistinguishable encryptions means that

$$\begin{gathered}\forall B, B' \;\; \overbrace{\Pr B'\big(E(K(1^n), B(1^n)_1), B(1^n)_1, B(1^n)_2, 1^n, B(1^n)_3\big)}^{\alpha_1 :\Leftrightarrow} - \\ \underbrace{\Pr B'\big(E(K(1^n), B(1^n)_2), B(1^n)_1, B(1^n)_2, 1^n, B(1^n)_3\big)}_{\Leftrightarrow: \alpha_2} \in \mathcal{N},\end{gathered} \tag{7.6}$$

where B and B' range over the polynomial-time probabilistic algorithms such that $(*)$ $\forall x \in \{0,1\}^*$ $|B(x)_1| = |B(x)_2|$.

First taking $(\dagger)$ $A(x) := (B(x)_1, B(x))$ and $(\dagger')$ $A'(x,y,z) := B'(x, z_1, z_2, y, z_3)$ in (7.5), and second taking $(\ddagger)$ $A(x) := (B(x)_2, B(x))$ and $(\ddagger')$ $A'(x,y,z) := B'(x, z_1, z_2, y, z_3)$ in (7.5), which are polynomial-time probabilistic algorithms because B and B' are polynomial-time probabilistic algorithms and $\cdot_1$, $\cdot_2$, $\cdot_3$ and $(\cdot,\cdot)$ are polynomial-time computable, we get, respectively,

$$\forall B, B' \ \overbrace{\Pr B'\big(E(K(1^n), B(1^n)_1), B(1^n)_1, B(1^n)_2, 1^n, B(1^n)_3\big)}^{\alpha_1 \Leftrightarrow} - \underbrace{\Pr B'(U_{|E(K(1^n),B(1^n)_1)|}, B(1^n)_1, B(1^n)_2, 1^n, B(1^n)_3)}_{\Leftrightarrow: \beta_1} \in \mathcal{N}, \tag{7.7}$$

$$\forall B, B' \ \overbrace{\Pr B'\big(E(K(1^n), B(1^n)_2), B(1^n)_1, B(1^n)_2, 1^n, B(1^n)_3\big)}^{\alpha_2 \Leftrightarrow} - \underbrace{\Pr B'(U_{|E(K(1^n),B(1^n)_2)|}, B(1^n)_1, B(1^n)_2, 1^n, B(1^n)_3)}_{\Leftrightarrow: \beta_2} \in \mathcal{N} \tag{7.8}$$

(notice that we made two different sets $(\dagger)$–$(\dagger')$ and $(\ddagger)$–$(\ddagger')$ of substitutions in (7.5), getting the two different formulas (7.7) and (7.8)). We have $|E(K(1^n), B(1^n)_1)| = |E(K(1^n), B(1^n)_2)|$ by $(*)$ and the length regularity of C, so $\beta_1 = \beta_2 =: \beta$, thus

$$\forall B, B' \ |\alpha_1 - \alpha_2| \leq \underbrace{|\alpha_1 - \beta|}_{\in \mathcal{N}} + \underbrace{|\alpha_2 - \beta|}_{\in \mathcal{N}} \in \mathcal{N},$$

hence we get (7.6).

7.30 Remark. The reciprocal implication of proposition 7.28 is false (assuming the existence of a length-regular stream cipher with indistinguishable encryptions): if $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ is a length-regular stream cipher with indistinguishable encryptions and $C' = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E', D')$ is the stream cipher defined by $E'(k,p) := E(k,p)1$ and $D'(k,c) := D(k, c_{\twoheadleftarrow})$, then C' is a length-regular stream cipher with indistinguishable encryptions but C' is not indistinguishable from random because the last bit $x_{\twoheadrightarrow}$ of an $x \in \{0,1\}^* \setminus \{\epsilon\}$ being 0 or 1 gives a clue about whether, respectively, x is not or is a ciphertext (the algorithms $A(x) := (0, \epsilon)$ and $A'(x,y,z) := x_{\twoheadrightarrow}$ are such that

$$\overbrace{\Pr A'\big(E'(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\big)}^{=1} - \underbrace{\Pr A'(U_{|E'(K(1^n),A(1^n)_1)|}, 1^n, A(1^n)_2)}_{=1/2} = 1/2 \notin \mathcal{N}).$$

7.31 Remark. Proposition 7.28 is false without the assumption that the stream cipher is length regular (assuming the existence of an indistinguishable-from-random length-regular stream cipher: if $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ is an indistinguishable-from-random length-regular stream cipher, then the non-length-regular stream cipher $C'' = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E'', D'')$ defined by $E''(k,p) := E(k, p[11 \leftarrow 1])$ and $D''(k,c) := D(k,c)[1 \leftarrow 11]$ (recall that $p[11 \leftarrow 1]$ means to replace all 1s in p by 11s, the intended meaning of $D(k,c)[1 \leftarrow 11]$ is to undo in $D(k,c)$ the said replacements, and so we have $\forall k \in \mathcal{K}\, \forall p \in \mathcal{P}\; D''(k, E''(k,p)) = p)$ is indistinguishable from random but does not have indistinguishable encryptions because $E''(k,0) = E(k, 0[11 \leftarrow 1]) = E(k,0)$ has length 1 but $E''(k,1) = E(k, 1[1 \leftarrow 11]) = E(k,11)$ has length 2 and so they so can be distinguished (the algorithms $A(x) := (0,1,\epsilon)$ and $A'(x,y,z) := \max(0, |x|-1)_2$ are such that

$$\overbrace{\Pr A'\big(E''(K(1^n), A(1^n)_1), A(1^n)_1, A(1^n)_2, 1^n, A(1^n)_3\big)}^{=0} -$$
$$\underbrace{\Pr A'\big(E''(K(1^n), A(1^n)_2), A(1^n)_1, A(1^n)_2, 1^n, A(1^n)_3\big)}_{=1} = -1 \notin \mathcal{N}).$$

7.8 Extra: semantic security

7.32. Let us prove that indistinguishability from random implies semantic security.

7.33 Proposition. For all length-regular stream ciphers C, if C is indistinguishable from random, then C is semantically secure.

7.34 Proof. Say $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$. The stream cipher C being indistinguishable from random means that

$$\begin{aligned} \forall A, A' \;\; &\Pr A'\big(E(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\big) - \\ &\Pr A'(U_{|E(K(1^n),A(1^n)_1)|}, 1^n, A(1^n)_2) \in \mathcal{N}, \end{aligned} \tag{7.9}$$

where A and A' range over the polynomial-time probabilistic algorithms. The stream cipher C being semantically secure means that

$$\begin{gathered} \forall B' \;\exists C' \;\forall B, f, g \\ \Pr\big[B'\big(E(K(1^n), B(1^n)_1), f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] - \\ \Pr\big[C'\big(1^{|B(1^n)_1|}, f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] \in \mathcal{N}, \end{gathered} \tag{7.10}$$

where B, B' and C' range over the polynomial-time probabilistic algorithms and f and g range over the polynomial-time computable functions.

Taking $A(x) := (B(x)_1, B(x))$ and $A'(x,y,z) := \mathrm{Tr}[B'(x, f(z,y), y, z_2) = g(z,y)]$, which are polynomial-time probabilistic algorithms because B and B' are polynomial-time probabilistic algorithms and $\cdot_1$, $\cdot_2$, $(\cdot,\cdot)$, f, g and $\mathrm{Tr}[\cdot = \cdot]$ are polynomial-time computable, in (7.9), we get

$$\begin{gathered} \forall B', B, f, g \\ \Pr \mathrm{Tr}\big[B'\big(E(K(1^n), B(1^n)_1), f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] - \\ \Pr \mathrm{Tr}\big[B'\big(U_{|E(K(1^n),B(1^n)_1)|}, f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] \in \mathcal{N}. \end{gathered}$$

Substituting $\Pr \mathrm{Tr}\, P$ by $\Pr P$, where P is any predicate, we get

$$\begin{gathered} \forall B', B, f, g \\ \Pr\big[B'\big(E(K(1^n), B(1^n)_1), f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] - \\ \Pr\big[B'\big(U_{|E(K(1^n),B(1^n)_1)|}, f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] \in \mathcal{N}. \end{gathered}$$

Substituting $(*)$ $|E(K(1^n), B(1^n)_1)|$ by $(\dagger)$ $|E(0, 1^{|B(1^n)_1|})|$, by the regularity of C, because the C' that we want to construct satisfying (7.10) does not have access to $(*)$ but can compute the equal $(\dagger)$ from the input $1^{|B(1^n)_1|}$, we get

$$\begin{gathered}\forall B', B, f, g\\ \Pr\bigl[B'\bigl(E(K(1^n), B(1^n)_1), f(B(1^n), 1^n), 1^n, B(1^n)_2\bigr) = g(B(1^n), 1^n)\bigr] -\\ \Pr\bigl[B'\bigl(U_{|E(0,1^{|B(1^n)_1|})|}, f(B(1^n), 1^n), 1^n, B(1^n)_2\bigr) = g(B(1^n), 1^n)\bigr] \in \mathcal{N}.\end{gathered}$$

Taking $C'(w, x, y, z) := B'(U_{|E(0,w)|}, x, y, z)$, we get (7.10).

7.35 Remark. The reciprocal implication of proposition 7.33 is false (assuming the existence of a semantically-secure length-regular stream cipher): if C is a semantically-secure length-regular stream cipher and C' is the stream cipher defined in remark 7.30, then C' is a semantically-secure length-regular stream cipher but C' is not indistinguishable from random as proved in remark 7.30.

7.36 Remark. Proposition 7.33 is false without the assumption that the stream cipher is length regular (assuming the existence of an indistinguishable-from-random length-regular stream cipher): if C is an indistinguishable-from-random length-regular stream cipher, then the non-length-regular stream cipher C'' defined in remark 7.31 is indistinguishable from random but it is not semantically secure because $|E''(k, 0)| = 1$ but $|E''(k, 1)| = 2$ and so, letting $p \in \{0, 1\}$, from $|E''(k, p)|$ we can compute p but from $1^{|p|} = 1$ we cannot compute p (the algorithms $A(x) := (U_1, \epsilon)$ and $A'(w, x, y, z) := \max(0, |w|-1)_2$ and the functions $f(x, y) := \epsilon$ and $g(x, y) := x_1$ are such that

$$\begin{gathered}\forall B\ \overbrace{\Pr\bigl[A'\bigl(E''(K(1^n), A(1^n)_1), f(A(1^n), 1^n), 1^n, A(1^n)_2\bigr) = g(A(1^n), 1^n)\bigr]}^{=1} -\\ \underbrace{\Pr\bigl[B\bigl(1^{|A'(1^n)_1|}, f(A(1^n), 1^n), 1^n, A(1^n)_2\bigr) = g(A(1^n), 1^n)\bigr]}_{=1/2} = 1/2 \notin \mathcal{N}).\end{gathered}$$

7.9 Extra: bit-recovery resistance

7.37. Let us prove that indistinguishability from random implies bit-recovery resistance.

7.38 Proposition. For all length-regular stream ciphers C, if C is indistinguishable from random, then C is bit-recovery resistant.

7.39 Proof. Say $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$. The stream cipher C being indistinguishable from random means that

$$\begin{gathered}\forall A, A'\ \Pr A'\bigl(E(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\bigr) -\\ \Pr A'(U_{|E(K(1^n),A(1^n)_1)|}, 1^n, A(1^n)_2) \in \mathcal{N},\end{gathered} \tag{7.11}$$

where A and A' range over the polynomial-time probabilistic algorithms. The stream cipher C being bit-recovery resistant means that

$$\forall B, B', i\ \Pr\bigl[B'\bigl(E(K(1^n), U_{|B(1^n)_1|}), 1^n, B(1^n)_2\bigr) = U^i_{|B(1^n)_1|}\bigr] - 1/2 \in \mathcal{N}, \tag{7.12}$$

where B and B' range over the polynomial-time probabilistic algorithms such that $\forall x \in \{0,1\}^* \; |B(x)_1| \geq 1$ and i ranges over $\mathbb{N}$.

Taking $A(x) := \big(U_{|B(x)_1|}, (B(x)_2, U^i_{|B(x)_1|})\big)$ and $A'(x,y,z) := \mathrm{Tr}[B'(x,y,z_1) = z_2]$, which are polynomial-time probabilistic algorithms because B and B' are polynomial-time probabilistic algorithms and $\cdot_1$, $\cdot_2$, $\cdot_3$, $(\cdot,\cdot)$, $|\cdot|$, $U_\cdot$, $\cdot^i$ and $\mathrm{Tr}[\cdot = \cdot]$ are polynomial-time computable, in (7.11), we get

$$\begin{aligned}\forall B, B', i \;\; &\Pr\mathrm{Tr}\big[B'\big(E(K(1^n), U_{|B(1^n)_1|}), 1^n, B(1^n)_2\big) = U^i_{|B(1^n)_1|}\big] - \\ &\Pr\mathrm{Tr}\big[B'(U_{|E(K(1^n),U_{|B(1^n)_1|})|}, 1^n, B(1^n)_2) = U^i_{|B(1^n)_1|}\big] \in \mathcal{N}.\end{aligned}$$

Substituting $\Pr\mathrm{Tr}\, P$ by $\Pr P$, where P is any predicate, we get

$$\begin{aligned}\forall B, B', i \; &\Pr\big[B'\big(E(K(1^n), U_{|B(1^n)_1|}), 1^n, B(1^n)_2\big) = U^i_{|B(1^n)_1|}\big] - \\ &\Pr\big[B'(U_{|E(K(1^n),U_{|B(1^n)_1|})|}, 1^n, B(1^n)_2) = U^i_{|B(1^n)_1|}\big] \in \mathcal{N}.\end{aligned}$$

We have $|E(K(1^n), U_{|B(1^n)_1|})| = |E(K(1^n), 1^{|B(1^n)_1|})|$ by the length regularity of C, so

$$\begin{aligned}\forall B, B', i \; &\Pr\big[B'\big(E(K(1^n), U_{|B(1^n)_1|}), 1^n, B(1^n)_2\big) = U^i_{|B(1^n)_1|}\big] - \\ &\underbrace{\Pr\big[B'(U_{|E(K(1^n),1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) = U^i_{|B(1^n)_1|}\big]}_{\Leftrightarrow:\alpha} \in \mathcal{N}.\end{aligned}$$

We have $\alpha = 1/2$ because $B'(U_{|E(K(1^n),1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2)$ and $U^i_{|B(1^n)_1|}$ are independent and $U^i_{|B(1^n)_1|}$ is a uniform random variable in $\{0,1\}$ since $|B(1^n)_1| \geq 1$, so we get (7.12).

7.40. It is worth remarking that the part "We have $\alpha = 1/2$ because $B'(U_{|E(K(1^n),1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2)$ and $U^i_{|B(1^n)_1|}$ are independent and $U^i_{|B(1^n)_1|}$ is a uniform random variable in $\{0,1\}$" of proof 7.39 uses a fact equivalent to the fact $(*)$ $U_n \perp X_n \;\Rightarrow\; U_n \oplus X_n = U'_n$ mentioned in paragraph 7.14: if the uniform random variable U_n in $\{0,1\}^n$ and a random variable X_n in $\{0,1\}^n$ are *independent*, then $\Pr[X_n = U_n] = 1/2^n$, or less precisely but more succinctly, $(\dagger)$ $X_n \perp U_n \;\Rightarrow\; \Pr[X_n = U_n] = 1/2^n$ (proof sketch:

$\underline{(\Rightarrow)}$ if $X_n \perp U_n$, then $X_n = U_n \;\Leftrightarrow\; U_n \oplus X_n = 0^n \;\Leftrightarrow\; U'_n = 0^n$ by $(*)$, so $\Pr[X_n = U_n] = \Pr[U'_n = 0^n] = 1/2^n$;

$\underline{(\Leftarrow)}$ if $U_n \perp X_n$, then for any constant $c_n \in \{0,1\}^n$, we have $U_n \oplus X_n = c_n \;\Leftrightarrow\; X_n \oplus c_n = U_n$, where $X'_n := X_n \oplus c_n \perp U_n$, so $\Pr[U_n \oplus X_n = c_n] = \Pr[X'_n = U_n] = 1/2^n$ by $(\dagger)$ with X'_n instead of X_n, thus $U_n \oplus X_n$ is uniform).

7.41 Remark. The reciprocal implication of proposition 7.38 is false (assuming the existence of a bit-recovery-resistant length-regular stream cipher): if C is a bit-recovery-resistant length-regular stream cipher and C' is the stream cipher defined in remark 7.30, then C' is a bit-recovery-resistant length-regular stream cipher but C' is not indistinguishable from random as proved in remark 7.30.

7.42 Remark. Proposition 7.38 is false without the assumption that the stream cipher is length regular (assuming the existence of an indistinguishable-from-random length-regular stream cipher): if C is an indistinguishable-from-random length-regular stream cipher, then the non-length-regular stream cipher C'' defined in remark 7.31 is indistinguishable from random but it is not bit-recovery resistant because we can recover $p \in \{0,1\}$ from $|E''(k,p)|$ as proved in remark 7.36 (the algorithms $A(x) := (0, \epsilon)$ and $A'(x,y,z) := \max(0, |x|-1)_2$ are such that

$$\underbrace{\Pr\left[A'\big(E''(K(1^n), U_{|A(1^n)_1|}), 1^n, A(1^n)_2\big) = U^i_{|A(1^n)_1|}\right]}_{=1} - 1/2 = 1/2 \notin \mathcal{N}).$$

7.10 Proof presentation: "compression" of analogous statements

7.43. We will improve the presentation of the claims of propositions 7.28, 7.33 and 7.38, remarks 7.30, 7.35 and 7.41, and also remarks 7.31, 7.36 and 7.42 (motivated by the reasons given in section 1.5). Although we are not working with a proof but with claims, we still call it proof presentation since the difference is of no importance.

7.44. Let us give, in the following table, names (on the left) to the sets of stream ciphers satisfying certain properties (on the right).

LR	Length Regularity
IFR	Indistinguishability From Random
IE	Indistinguishable Encryptions
SS	Semantic Security
BRR	Bit-Recovery Resistance

Given a stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$, let us recall, in the following table, the stream ciphers $C' = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E', D')$ and $C'' = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E'', D'')$ from remarks 7.30 and 7.31.

C'	$E'(k,p) := E(k,p)1$	$D'(k,c) := D(k, c_{\twoheadleftarrow})$
C''	$E''(k,p) := E(k, p[11 \leftarrow 1])$	$D''(k,c) := D(k,c)[1 \leftarrow 11]$

Then the mentioned propositions and remarks can be neatly "compressed" into the following table.

Proposition 7.28	$C \in \mathrm{LR} \cap \mathrm{IFR}$	$\Rightarrow$	$C \in \mathrm{LR} \cap \mathrm{IE}$
Remark 7.30	$C \in \mathrm{LR} \cap \mathrm{IE}$	$\Rightarrow$	$C' \in (\mathrm{LR} \cap \mathrm{IE}) \setminus \mathrm{IFR}$
Remark 7.31	$C \in \mathrm{LR} \cap \mathrm{IFR}$	$\Rightarrow$	$C'' \in \mathrm{IFR} \setminus (\mathrm{LR} \cup \mathrm{IE})$
Proposition 7.33	$C \in \mathrm{LR} \cap \mathrm{IFR}$	$\Rightarrow$	$C \in \mathrm{LR} \cap \mathrm{SS}$
Remark 7.35	$C \in \mathrm{LR} \cap \mathrm{SS}$	$\Rightarrow$	$C' \in (\mathrm{LR} \cap \mathrm{SS}) \setminus \mathrm{IFR}$
Remark 7.36	$C \in \mathrm{LR} \cap \mathrm{IFR}$	$\Rightarrow$	$C'' \in \mathrm{IFR} \setminus (\mathrm{LR} \cup \mathrm{SS})$
Proposition 7.38	$C \in \mathrm{LR} \cap \mathrm{IFR}$	$\Rightarrow$	$C \in \mathrm{LR} \cap \mathrm{BRR}$
Remark 7.41	$C \in \mathrm{LR} \cap \mathrm{BRR}$	$\Rightarrow$	$C' \in (\mathrm{LR} \cap \mathrm{BRR}) \setminus \mathrm{IFR}$
Remark 7.42	$C \in \mathrm{LR} \cap \mathrm{IFR}$	$\Rightarrow$	$C'' \in \mathrm{IFR} \setminus (\mathrm{LR} \cup \mathrm{BRR})$

Let X range over $\{\mathrm{IE}, \mathrm{SS}, \mathrm{BRR}\}$. Then the previous table can be further "compressed" into the following table by taking advantage of the analogous structures of the propositions and remarks.

Propositions 7.28, 7.33, 7.38	$C \in \mathrm{LR} \cap \mathrm{IFR} \Rightarrow C \in \mathrm{LR} \cap X$
Remarks 7.30, 7.35, 7.41	$C \in \mathrm{LR} \cap X \Rightarrow C' \in (\mathrm{LR} \cap X) \setminus \mathrm{IFR}$
Remarks 7.31, 7.36, 7.42	$C \in \mathrm{LR} \cap \mathrm{IFR} \Rightarrow C'' \in \mathrm{IFR} \setminus (\mathrm{LR} \cup X)$

7.45. We see two advantages in the last table over the original claim texts of the propositions and the remarks:

1. the table compresses all claim texts in only three lines;
2. the table makes it clear that the propositions and remarks are analogous to each other by showing that they are instantiations (by instantiating X) of a master structure (the table with the variable X).

7.11 Conclusion

7.46. In this chapter:

1. we solved the instance of our problem specified by

$$\begin{aligned} P_1 &= \text{pseudorandom generator}, \\ S_1 &= \text{cryptographic security}, \\ P_2 &= \text{stream cipher}, \\ S_2 &= \text{indistinguishability from random}, \\ T &= \text{transformation of a pseudorandom generator} \\ & \text{into its induced stream cipher}; \end{aligned}$$

2. we gave three proof presentations
   (a) using a schematic proof;
   (b) using the wedding-cake notation;
   (c) by "compressing" together the claims of analogous propositions and remarks;
3. we proved that indistinguishability from random implies
   (a) cryptographic security;
   (b) indistinguishable encryptions;
   (c) semantic security;
   (d) bit-recovery resistance.

Chapter 8

Transforming one-way binary-string functions into $\mathrm{NP} \setminus \mathrm{P}$ formal languages

8.1 Introduction

8.1. In this chapter we solve the instance of our problem format (presented in paragraph 1.28) specified by the following instances of its parameters:

$$\begin{aligned}
P_1 &= \text{binary-string function,}\\
S_1 &= \text{one-wayness,}\\
P_2 &= \text{formal language,}\\
S_2 &= (\mathrm{NP} \setminus \mathrm{P})\text{-ness,}\\
T &= \text{transformation of a binary-string function}\\
&\quad\ \text{into its induced bitwise formal language.}
\end{aligned}$$

As a proof presentation, we "carve out" from the security proof of T a "mini-theory" of minimisation operators and use it to reprove the security of T in a neater way.

8.2. Considering a formal language as a cryptographic primitive P_2 and $(\mathrm{NP} \setminus \mathrm{P})$-ness as a cryptographic security notion S_2 is to some extent artificial: they are more related to computation theory and complexity theory than to cryptography but they do play an essential role in arguing the difficulty of presenting an example of a one-way binary-string function, which belongs to cryptography; it is in this broader sense that we talk about the cryptographic primitive P_2 and its security notion S_2.

8.3. Let us informally explain our instances of the parameters.

Binary-string function It is a function that inputs and outputs binary strings.

One-wayness It means that the function f is easy to compute but hard to invert, that is $x \mapsto f(x)$ is easy to compute but $f(x) \mapsto x$ (or more precisely, $f(x) \mapsto x'$ with $f(x) = f(x')$) is hard to compute.

Formal language It is a set of words such as $\{0, 01, 011, 0111, \ldots\}$ written with letters such as 0 and 1 from some alphabet such as $\{0, 1\}$.

NP \ P It means that the problem of deciding if a word belongs to the formal language is a hard problem.

Transformation It inputs a binary-string function f and outputs a formal language capturing the hardness of inverting f.

8.4. In this chapter all work is ours except that:

1. the theorem stating that the existence of one-way functions implies P $\neq$ NP and its informal "proof" are mostly unpublished "folklore";
2. we adopted in our proof a simplification by Aaron Dutle of his less formal proof (Dutle et al. 2009, problem 2);
3. the analogy in paragraph 8.16 is due to Eerke Boiten.

8.2 Transformation

8.5. The problem that we solve here can be briefly stated as "one-way functions imply P $\neq$ NP" (this statement is less informative because it abstracts the transformation T). It is a "folklore" result that explains why there are no known unconditional examples of one-way functions (since it is unknown whether P $\neq$ NP or not). The idea to prove it is simple: if f is a one-way binary-string function, then the problem of inverting f is in NP \ P (this idea is imprecise because NP \ P is a class of formal languages and not problems). Despite this, it is difficult to find a detailed and correct proof in the literature. The proofs that we found suffer from the following problems.

Mistakes Some proofs construct a language $\{x2f(\bar{x}) \mid x \sqsubseteq \bar{x} \in \{0,1\}^*\}$ (similar to $\bar{L}_f$ in definition 8.8) which causes a certain algorithm $B(x2f(\bar{x}))$ (similar to B in proof 8.13) not to run in polynomial-time (essentially because from $|f(\bar{x})|$ we may not recover $|\bar{x}|$).

Informality Some proofs present the informal idea above without formalising it enough to allow to check the proof for correctness.

We are only aware of two exceptions (proofs without mistakes and formal enough):

1. a proof by Aaron Dutle (Dutle et al. 2009, problem 2);
2. our own initial proof.

These two proofs are similar but have the following differences.

	Dutle	Gaspar
$\{x2f(\bar{x})2^n \mid x \sqsubseteq \bar{x} \in \{0,1\}^*, \ldots\}$	$\ldots n = \lvert\bar{x}\rvert$	$\ldots n \geq \lvert\bar{x}\rvert$
B searches for an x with…	$\ldots n = \lvert\bar{x}\rvert$	$\ldots n \geq \lvert\bar{x}\rvert$
The proof is more…	… informal	… formal

The choice $n = |\bar{x}|$ in Dutle's proof simplifies the proof when compared to our proof, so below we adopt Dutle's choice.

8.6. Below we will consider formal languages over the alphabet $\{0, 1, 2\}$. We will treat the letter 2 as a separator between binary strings. For example 2 is used in $x2y2z$ (where $x, y, z \in \{0, 1\}^*$) to separate x, y and z. We can think of:

1. 2 as a comma ",";
2. $x2y2z$ as "x, y, z".

8.7. In theorem 8.12 we will show that a one-way binary-string function f can be transformed into an NP $\setminus$ P formal language $\bar{L}_f$, so now we define $\bar{L}_f$. We also define a simplified variant L_f of $\bar{L}_f$ whose role will become clear in section 8.4.

8.8 Definition. Let f be a binary-string function.

1. The *formal language* L_f *induced by* f is $\{x2f(x)21^{|x|} \mid x \in \{0, 1\}^*\}$.
2. The *bitwise formal language* $\bar{L}_f$ *induced by* f is $\{x2f(\bar{x})21^{|\bar{x}|} \mid x \sqsubseteq \bar{x} \in \{0, 1\}^*\}$.

8.9. Now, to be sure, we explicitly state the transformation in question.

8.10 Definition. The *transformation* of a binary-string function f into its induced bitwise formal language $\bar{L}_f$ is $f \rightsquigarrow \bar{L}_f$.

8.3 Security

8.11. Now we show that if a binary-string function is one-way, then its induced bitwise formal language is NP $\setminus$ P. Informally, this means that secure (in some sense) binary-string functions are transformed into secure (also in some sense) formal languages.

8.12 Theorem. For all binary-string functions f, if f is one-way, then $\bar{L}_f$ is NP $\setminus$ P.

8.13 Proof. The structure of the proof is the following:

1. we prove that $\bar{L}_f$ is NP by
   (a) constructing a nondeterministic algorithm B;
   (b) proving that B decides $\bar{L}_f$;
   (c) proving that B runs in polynomial time;
2. we prove $\bar{L}_f$ is not P by
   (a) constructing from a hypothetical polynomial-time deterministic algorithm C deciding $\bar{L}_f$ a deterministic algorithm D;
   (b) proving that D inverts f;
   (c) proving that D runs in polynomial time;

contradicting the one-wayness of f.

So let us do this.

$\bar{L}_f$ is NP

Construction of B Let us consider the following nondeterministic algorithm B, described in high-level pseudo-code, that inputs $w \in \{0,1,2\}^*$ and outputs `yes` or `no`. In the first line below, B checks that w has the right format $x2y2z$ (otherwise returns the "error message" `no` and halts) and parses w, and in the third line B uses the parsed x, y and z.

if not $\underbrace{\exists x, y \in \{0,1\}^{\leq |w|}\ \exists z \in \{1\}^{\leq |w|}\ w = x2y2z}_{(1)}$
then {output `no`; halt}
if $\underbrace{\exists \bar{x} \in \{0,1\}^{|z|}\ (\overbrace{x \sqsubseteq \bar{x} \wedge y = f(\bar{x})}^{(2)})}_{(3)}$
then {output `yes`; halt}
output `no`; halt

In order to determine below the runtime of B, we need to give some low-level detail about how B checks (1) and (3).

(1) To check (1), B counts the number of 2s in w and the number of 0s after the second 2 (if there is a second 2), determines that (1) is true if the counted number of 2s is two and the counted number of 0s is zero (so there are only 1s to the right of the second 2), otherwise determines that (1) is false.
Let us notice that B will determine that (1) is true exactly when w has exactly two 2s and only 1s to the right of the second 2, that is exactly when (1) is indeed true.

(3) To check (3), B uses its "internal coin tosses" to construct a random $\bar{x} \in \{0,1\}^{|z|}$, checks whether (2) is true and determines that (3) is true if (2) is true, otherwise determines that (3) is false.
Let us notice that B, being nondeterministic, will determine that (3) is true exactly when there is a construction of an $\bar{x} \in \{0,1\}^{|z|}$ such that (2) is true, that is exactly when (3) is indeed true.

B decides $\bar{L}_f$ Let us prove $w \in \bar{L}_f \;\Leftrightarrow\; B(w) = \texttt{yes}$ (for all $w \in \{0,1,2\}^*$): $w \in \bar{L}_f$ if and only if $w = x2f(\bar{x})21^{|\bar{x}|}$ for some $x, \bar{x} \in \{0,1\}^*$ with $x \sqsubseteq \bar{x}$, which is equivalent to (1) and (3) being both true, that is $B(w) = \texttt{yes}$.

B runs in polynomial time It essentially suffices to prove that (1), (2) and (3) are checked in polynomial time.

(1) It should be clear that B can count the number of 2s in w and the number of 0s in w after the second 2 in polynomial time in $|w|$.

(2) Checking (2) takes polynomial-time in $|\bar{x}| = |z| \leq |w|$ because $\sqsubseteq$ is polynomial-time decidable and f is polynomial-time computable.

(3) The construction of $\bar{x} \in \{0,1\}^{|z|}$ takes $|\bar{x}| = |z| \leq |w|$ steps and (as already seen) checking (2) takes polynomial-time in $|w|$.

$\bar{L}_f$ is not P Let us assume, aiming at a contradiction, that $\bar{L}_f$ is P, that is there is a polynomial-time deterministic algorithm C deciding $\bar{L}_f$.

Construction of D Let us consider the following deterministic algorithm D, described in high-level pseudo-code, that inputs $w \in \{0,1,2\}^*$ and outputs `no` or a certain $x_{|z|} \in \{0,1\}^*$. In the first line below, D checks that w has the right format $y2z$ (otherwise returns the "error message" `no` and halts) and parses w, and in the third to seventh lines D uses the parsed y and z and constructs $x_{|z|}$ bitwise (one bit at a time).

$$
\begin{array}{l}
\text{if not } \underbrace{\exists y \in \{0,1\}^{\leq |w|}\ \exists z \in \{1\}^{\leq |w|}\ w = y2z}_{(4)} \\
\text{then } \{\text{output } \texttt{no};\ \text{halt}\} \\
i := 0;\ x_0 := \epsilon \\
\left.\begin{array}{l}
\text{for } i \text{ from } 1 \text{ to } |z| \\
\left.\begin{array}{l}
\text{if } \underbrace{C(x_{i-1}02y2z) = \texttt{yes}}_{(5)} \\
\text{then } x_i := x_{i-1}0 \\
\text{else } x_i := x_{i-1}1
\end{array}\right\} (6)
\end{array}\right\} (7) \\
\text{output } x_{|z|};\ \text{halt}
\end{array}
$$

In order to determine below the runtime of D, we would need to give some low-level detail about how D checks (4), but instead we just mention that is analogous to how B checks (1).

D inverts f Let $x \in \{0,1\}^*$ (which induces $(y,z) := (f(x), 1^{|x|})$). Let us denote by $x_{\min}$ the least (with respect to the lexicographic order) inverse of $f(x)$ (that is y) with length $|x|$ (that is $|z|$), that is $x_{\min} := \min_{\text{lex}} X$, where $X := f^{-1}[f(x)] \cap \{0,1\}^{|x|}$ (that is $X := f^{-1}[y] \cap \{0,1\}^{|z|}$). Let us prove that D inverts f by proving the following: if D inputs $w = f(x)21^{|x|}$, then D outputs $x_{|z|} = x_{\min}$. So let us assume that D inputs $w = f(x)21^{|x|}$. Then (4) is true with $y := f(x)$ and $z := 1^{|x|}$ (and y and z such that (4) are unique), so D proceeds to the construction of $x_0, \ldots, x_{|z|}$. We can prove $|x_i| = i$ by a simple induction on $i \in [0\,..\,|z|]$ by noticing $x_0 = \epsilon$ and that in (6) each x_i is obtained by appending a single 0 or 1 to x_{i-1}. Let us prove $x_i \sqsubseteq x_{\min}$ by induction on $i \in [0\,..\,|z|]$.

Base case We have $x_0 = \epsilon$, so $x_0 \sqsubseteq x_{\min}$.

Induction step We assume $x_{i-1} \sqsubseteq x_{\min}$ by induction hypothesis, we have $|x_{i-1}| < |x_{\min}|$ because $|x_{i-1}| = i - 1 < |z| = |x| = |x_{\min}|$, so $x_{i-1}0 \sqsubseteq x_{\min} \vee x_{i-1}1 \sqsubseteq x_{\min}$. We have

$$
(5) \Leftrightarrow x_{i-1}02y2z \in \bar{L}_f \Leftrightarrow
$$
$$
\exists \bar{x} \in \{0,1\}^* \ (x_{i-1}0 \sqsubseteq \bar{x} \wedge y = f(\bar{x}) \wedge |\bar{x}| = |z|) \Leftrightarrow x_{i-1}0 \sqsubseteq x_{\min}
$$

because $x_{i-1} \sqsubseteq x_{\min}$, $\bar{x}$ is an inverse of $y = f(x)$ with length $|\bar{x}| = |z| = |x|$ and $x_{\min}$ is the minimum of such inverses, so (interpreting

(6) as a predicate instead of a statement)

$$(6) \Leftrightarrow$$
$$(x_{i-1}0 \sqsubseteq x_{\min} \Rightarrow x_i := x_{i-1}0) \wedge (x_{i-1}1 \sqsubseteq x_{\min} \Rightarrow x_i := x_{i-1}1)$$

because $x_{i-1}0 \sqsubseteq x_{\min} \vee x_{i-1}1 \sqsubseteq x_{\min}$, thus $x_i \sqsubseteq x_{\min}$.

Taking $i = |z|$ in $|x_i| = i$ and $x_i \sqsubseteq x_{\min}$, we get $x_{|z|} = x_{\min}$ because $|z| = |x_{\min}|$.

<u>D runs in polynomial time</u> It essentially suffices to prove that (4), (5) and (7) run in polynomial time.

<u>(4)</u> Checking (4) takes polynomial time in $|w|$ analogously to (1).

<u>(5)</u> Checking (5) takes polynomial time in $|x_{i-1}02y2z| \leq 2|w|$ because C runs in polynomial time, $w = y2z$ and $|x_{i-1}02| \leq |z| + 1 \leq |w|$ since $|x_{i-1}| = i - 1 < |z|$.

<u>(7)</u> Running (7) takes polynomial time in $|w|$ because the for loop runs through $|z| \leq |w|$ values and the instructions and checks inside the loop, notably C in (5), run in polynomial time in $|w|$.

8.4 Proof presentation: carving out a theory

8.14. We will improve the proof presentation of proof 9.10 (motivated by the reasons given in section 1.5).

8.15. In (7) in proof 8.13 there is a bitwise construction $x_0 \rightsquigarrow \cdots \rightsquigarrow x_{|z|}$ of an inverse $x_{|z|}$ of $f(x)$. Now we work on a proof presentation of proof 8.13 by "carving out" this construction into a "mini-theory" of minimisation operators μ and $\bar{\mu}$.

8.16. To illustrate this "mini-theory", let us consider the following analogy suggested by Eerke Boiten. Let us say that a thief wants to open a safe with a three-digit mechanical combination lock, whose combination is 123.

1. Let $L := \{123\}$. If the thief is stupid, then he/she will try all combinations from 000 to 999 until one works. We can think that the thief is trying to find an x in L by running through all possible elements x and testing membership in L until he/she finds an x such that $x \in L$. Notice that this (in the worst-case scenario) requires 10^3 trials (in general it would be 10^n where n is the number of digits), which is exponential in the number of digits.

2. Let $\bar{L} := \{x \mid x \sqsubseteq \bar{x} \in L\} = \{\epsilon, 1, 12, 123\}$. If the thief is smart, then he/she will search for the first digit by listening to the lock for a click that signals the first digit $d_1 := 1$, and then he/she will similarly find the second and third digits $d_2 := 2$ and $d_3 := 3$. We can think that the thief is trying to find an x_3 in L by constructing the finite sequence

$$\begin{aligned}
&x_0 := \epsilon, \\
&x_1 := x_0 d_1 = 1, \quad \text{where } d_1 \in [0\,..\,9] \text{ is such that } x_0 d_1 \in \bar{L}, \\
&x_2 := x_1 d_2 = 12, \quad \text{where } d_2 \in [0\,..\,9] \text{ is such that } x_1 d_2 \in \bar{L}, \\
&x_3 := x_2 d_3 = 123, \text{ where } d_3 \in [0\,..\,9] \text{ is such that } x_2 d_3 \in \bar{L}
\end{aligned}$$

by testing membership in $\bar{L}$. Notice that this (in the worst-case scenario) requires 3×10 trials (in general it would be $n \times 3$ where n is the number of digits), which is linear in the number of digits.

8.17. Below we will consider a formal language L over the alphabet $\{0, 1, 2\}$ and a formula $\exists x \in \{0, 1\}^{|z|}\ x2y2z \in L$. We will treat:

1. y as a parameter (which can be ϵ if we do not want a parameter, and which can "morally" be finitely many binary parameters $y_1, \ldots, y_n$ by encoding $y_1, \ldots, y_n$ as a single binary parameter y);

2. z as another parameter, of the form 1^l (where $l \in \mathbb{N}$), which we will only use to give us the length $l = |z| = |x|$ of the x in the formula.

To simplify the discussion that follows, we will drop 2 and y and replace z by l in paragraph 8.18. Then, to be rigorous, we will resume using them in definitions 8.21 and 8.23.

8.18. Let L be a formal language over the alphabet $\{0, 1\}$ and let us consider the formula $\exists x \in \{0, 1\}^n\ x \in L$. We will define two operators μ and $\bar{\mu}$ that search for the least witness x to this formula.

1. The operator μ is $\mu(L) := \min_{\text{lex}} X$, where $X := \{x \in \{0, 1\}^n \mid x \in L\} = \{0, 1\}^n \cap L$. Notice that $\mu(L)$ is a witness to the formula (if there is a witness). The deterministic algorithm that we have in mind to compute $\mu(L)$ is simply a brute-force search: we run though all possible $x \in \{0, 1\}^n$ in lexicographic order and test for $x \in L$ until we find such an x. Notice that this algorithm runs in exponential time in n (provided that we can test membership in $\bar{L}$ in at most exponential time).

Let $\bar{L} := \{x \mid x \sqsubseteq \bar{x} \in \{0, 1\}^n,\ \bar{x} \in L\}$. This formal language is the "prefix-closure" of L (more precisely, of X).

2. The operator $\bar{\mu}$ is $\bar{\mu}(\bar{L}) := x_n$, where the finite sequence $(x_i)_{i=0,\ldots,n}$ is recursively defined by
$$\begin{cases} x_0 := \epsilon & \text{if } \epsilon \in \bar{L} \\ x_i := \begin{cases} x_{i-1}0 & \text{if } x_{i-1}0 \in \bar{L} \\ x_{i-1}1 & \text{if } x_{i-1}1 \in \bar{L} \end{cases} \end{cases}$$
(this unofficial definition has the problem that we may not have "$\epsilon \in \bar{L}$" or "$x_{i-1}0 \in \bar{L}$ or $x_{i-1}1 \in \bar{L}$ but not both", but this is taken care of in the official definition 8.23 below). Although it may not be obvious, $\bar{\mu}(L)$ is the least witness to the formula, which is $\mu(L)$. The algorithm that we have in mind to compute $\bar{\mu}(L)$ is the construction of $(x_i)_{i=0,\ldots,n}$ following its recursive definition. Notice that this algorithm runs in polynomial time in n (provided that we can test membership in $\bar{L}$ in at most polynomial time).

8.19. It may seem that what $\mu(L)$ does in exponential time is done by $\bar{\mu}(\bar{L})$ in polynomial-time, which by "magic" reduces the complexity from exponential to linear. But the complexity does not really disappear: it is simply shifted from μ to $\bar{L}$, in the sense of the complexity decreasing in $\mu \rightsquigarrow \bar{\mu}$ (informally, from NP to P) but potentially increasing in $L \rightsquigarrow \bar{L}$ (formally, from P to NP), as schematised by the following table.

$\mu(L)$		$\bar{\mu}(\bar{L})$	
μ	L	$\bar{\mu}$	$\bar{L}$
NP	P	P	NP

8.20. Now we formally define $\bar{L}$, μ and $\bar{\mu}$. The definition of $\bar{\mu}$ (more precisely, of $(x_i)_{i=0,\ldots,|z|}$) is more complicated than what we sketch in paragraph 8.18 to deal with the problems about possibly not having "$\epsilon \in \bar{L}$" or "$x_{i-1}0 \in \bar{L}$ or $x_{i-1}1 \in \bar{L}$ but not both" mentioned there.

8.21 Definition. The *bitwise formal language* $\bar{L}$ *induced by the formal language* $L \subseteq \{0,1,2\}^*$ *is*

$$\{x2y2z \mid x \sqsubseteq \bar{x} \in \{0,1\}^{|z|},\ z \in \{1\}^*,\ \bar{x}2y2z \in L\}.$$

8.22. The notation $\bar{L}_f$ may seem ambiguous because it can denote

1. the language $\bar{L}_f$ in definition 8.8;
2. language $\bar{L}$ in definition 8.21 with $L := L_f$;

but these two languages coincide so there is really no ambiguity.

8.23 Definition.

1. The *minimisation operator* μ is the partial function

$$\begin{aligned} \mu\colon \mathcal{P}(\{0,1,2\}^*) \times \{0,1\}^* \times \{1\}^* &\nrightarrow \{0,1\}^* \\ (L,y,z) &\mapsto \min\nolimits_{\mathrm{lex}} X \end{aligned},$$

   where

$$X := \{x \in \{0,1\}^{|z|} \mid x2y2z \in L\}.$$

2. The *bitwise minimisation operator* $\bar{\mu}$ is the partial function

$$\begin{aligned} \bar{\mu}\colon \mathcal{P}(\{0,1,2\}^*) \times \{0,1\}^* \times \{1\}^* &\nrightarrow \{0,1\}^* \\ (L,y,z) &\mapsto x_{|z|} \end{aligned},$$

   where the finite sequence $(x_i)_{i=0,\ldots,|z|} \subseteq \{0,1\}^*$ is recursively defined by

$$\begin{cases} x_0 := \begin{cases} \epsilon & \text{if } \epsilon 2y2z \in L \\ \uparrow & \text{otherwise} \end{cases} \\ x_i := \begin{cases} x_{i-1}0 & \text{if } x_{i-1}02y2z \in L \\ x_{i-1}1 & \text{if } x_{i-1}02y2z \notin L \wedge x_{i-1}12y2z \in L \\ \uparrow & \text{otherwise} \end{cases} \end{cases}.$$

8.24. If $\bar{\mu}$ is applied to a language of the form $\bar{L}$ instead of an arbitrary language L, then the definition of x_i can be simplified by removing the case "otherwise" because we have $\epsilon 2y2z \in \bar{L} \;\Rightarrow\; x_{i-1}02y2z \in \bar{L} \vee x_{i-1}12y2z \in \bar{L}$.

8.25. In the next proposition we show that $L \rightsquigarrow \bar{L}$ potentially increases the complexity from P to NP as discussed in paragraph 8.19.

8.26 Proposition. For all $L \in \mathcal{P}(\{0,1,2\}^*)$, if L is P, then $\bar{L}$ is NP.

8.27 Proof. The structure of the proof is the following: we assume that L is P, that is there is a polynomial-time deterministic algorithm A deciding L, and we prove that $\bar{L}$ is NP by:

1. constructing a nondeterministic algorithm B;
2. proving that B decides $\bar{L}$;
3. proving that B runs in polynomial time.

So let us do this.

Construction of B Let us consider the following nondeterministic algorithm B, described in high-level pseudo-code, that inputs $w \in \{0,1,2\}^*$ and outputs `yes` or `no`. In the first line below, B checks that w has the right format $x2y2z$ (otherwise returns the "error message" `no` and halts) and parses w, and in the third line B uses the parsed x, y and z.

$$\begin{array}{l}
\text{if not } \underbrace{\exists x, y \in \{0,1\}^{\leq |w|}\ \exists z \in \{1\}^{\leq |w|}\ w = x2y2z}_{(1)} \\
\text{then } \{\text{output } \texttt{no};\ \text{halt}\} \\
\text{if } \underbrace{\exists \bar{x} \in \{0,1\}^{|z|}\ \overbrace{(x \sqsubseteq \bar{x} \wedge A(\bar{x}2y2z) = \texttt{yes})}^{(2)}}_{(3)} \\
\text{then } \{\text{output } \texttt{yes};\ \text{halt}\} \\
\text{output } \texttt{no};\ \text{halt}
\end{array}$$

In order to determine below the runtime of B, we would need to give some low-level detail about how B checks (1) and (3): this is as in proof 8.13 (in particular, B, being nondeterministic, can use its "internal coin tosses" to construct a random $\bar{x} \in \{0,1\}^{|z|}$ in linear time in $|z|$).

B decides $\bar{L}$ Let us prove $w \in \bar{L} \Leftrightarrow B(w) = \texttt{yes}$ (for all $w \in \{0,1,2\}^*$): $w \in \bar{L}$ if and only if $w = x2y2z$ for some $x, \bar{x}, y \in \{0,1\}^*$ and $z \in \{1\}^*$ with $x \sqsubseteq \bar{x} \in \{0,1\}^{|z|}$ and $\bar{x}2y2z \in L$, which is equivalent to (1) and (3) being both true because A decides L, that is $B(w) = \texttt{yes}$.

B runs in polynomial time It essentially suffices to prove that (1), (2) and (3) are checked in polynomial time.

(1) This is as in proof 8.13.

(2) Checking (2) takes polynomial-time in $|\bar{x}2y2z| \leq 2|w|$ because $\sqsubseteq$ is polynomial-time decidable, A runs in polynomial-time, $|\bar{x}| = |z| \leq |w|$ and $|2y2z| \leq |x2y2z| = |w|$.

(3) This is as in proof 8.13.

8.28. In the next proposition we show (informally speaking) $\mu(L, y, z) = \bar{\mu}(\bar{L}, y, z)$ as discussed in paragraph 8.18.

8.29 Proposition. For all $L \in \mathcal{P}(\{0,1,2\}^*)$, $x, y \in \{0,1\}^*$ and $z \in \{1\}^*$, we have $\mu(L, y, z) \overset{\downarrow\uparrow}{=} \bar{\mu}(\bar{L}, y, z)$.

8.30 Proof. Let X and (x_i) be as in definition 8.23.

$\mu(L, y, z)\uparrow \;\Rightarrow\; \bar{\mu}(\bar{L}, y, z)\uparrow$ If $\mu(L, y, z)\uparrow$, then $X = \emptyset$, so $\nexists x \in \{0,1\}^{|z|}\; x2y2z \in L$, thus $\epsilon 2y2z \notin \bar{L}$, hence $x_0\uparrow$, therefore $x_{|z|}\uparrow$, so $\bar{\mu}(\bar{L}, y, z)\uparrow$.

$\mu(L, y, z)\downarrow \;\Rightarrow\; \bar{\mu}(\bar{L}, y, z)\downarrow \wedge \mu(L, y, z) = \bar{\mu}(\bar{L}, y, z)$ Let us assume $\mu(L, y, z)\downarrow$, that is $X \neq \emptyset$. Let us denote by $x_{\min}$ the least (with respect to the lexicographic order) $x \in \{0,1\}^{|z|}$ such that $x2y2z \in L$, that is $x_{\min} := \min_{\text{lex}} X$. Let us prove $\bar{\mu}(\bar{L}, y, z)\downarrow$ and $\mu(L, y, z) = \bar{\mu}(\bar{L}, y, z)$ by proving $x_{|z|}\downarrow$ and $x_{|z|} = x_{\min}$. We can prove $x_i\downarrow \;\Rightarrow\; |x_i| = i$ by a simple induction on $i \in [0 \mathinner{..} |z|]$ by noticing $x_0\downarrow \;\Rightarrow\; x_0 = \epsilon$ and that each x_i, if defined, is obtained by appending a single 0 or 1 to x_{i-1}. Let us prove $x_i\downarrow \wedge x_i \sqsubseteq x_{\min}$ by induction on $i \in [0 \mathinner{..} |z|]$.

Base case We have $x_0 = \epsilon$ because $\epsilon 2y2z \in \bar{L}$ since $\epsilon \sqsubseteq x_{\min}$ and $x_{\min}2y2z \in L$, so $x_0\downarrow \wedge x_0 \sqsubseteq x_{\min}$.

Induction step We assume $x_{i-1}\downarrow \wedge x_{i-1} \sqsubseteq x_{\min}$ by induction hypothesis, we have $|x_{i-1}| < |x_{\min}|$ because $|x_{i-1}| = i-1 < |z| = |x_{\min}|$, so $x_{i-1}0 \sqsubseteq x_{\min} \vee x_{i-1}1 \sqsubseteq x_{\min}$. We have

$$x_{i-1}02y2z \in \bar{L} \;\Leftrightarrow$$
$$\exists \bar{x} \in \{0,1\}^* \; (x_{i-1}0 \sqsubseteq \bar{x} \wedge \bar{x}2y2z \in L \wedge |\bar{x}| = |z|) \;\Leftrightarrow\; x_{i-1}0 \sqsubseteq x_{\min}$$

because $x_{i-1} \sqsubseteq x_{\min}$, $\bar{x}$ is such that $\bar{x}2y2z \in L$ and $|\bar{x}| = |z|$, and $x_{\min}$ is the least of such $\bar{x}$s, and analogously

$$x_{i-1}02y2z \notin \bar{L} \wedge x_{i-1}12y2z \in \bar{L} \;\Leftrightarrow\; x_{i-1}1 \sqsubseteq x_{\min},$$

so

$$x_i := \begin{cases} x_{i-1}0 & \text{if } x_{i-1}02y2z \in \bar{L} \\ x_{i-1}1 & \text{if } x_{i-1}02y2z \notin \bar{L} \wedge x_{i-1}12y2z \in \bar{L} \\ \uparrow & \text{otherwise} \end{cases}$$
$$\Leftrightarrow\; (x_{i-1}0 \sqsubseteq x_{\min} \;\Rightarrow\; x_i := x_{i-1}0) \wedge (x_{i-1}1 \sqsubseteq x_{\min} \;\Rightarrow\; x_i := x_{i-1}1),$$

thus $x_i \sqsubseteq x_{\min}$.

Taking $i = |z|$ in $x_i\downarrow$, $|x_i| = i$ and $x_i \sqsubseteq x_{\min}$, we get $x_{|z|}\downarrow$ and $x_{|z|} = x_{\min}$ because $|z| = |x_{\min}|$.

8.31 Theorem. For all binary-string functions f, if f is one-way, then $\bar{L}_f$ is NP $\setminus$ P.

8.32 Proof.

$\bar{L}_f$ is NP We have that L_f is P because $\cdot 2 \cdot 2 \cdot$, $1^{\cdot}$, $|\cdot|$ and f are polynomial-time computable, so $\bar{L}_f$ is NP by proposition 8.26.

$\bar{L}_f$ is not P Let us assume, aiming at a contradiction, that $\bar{L}_f$ is P. Let us construct a deterministic polynomial-time algorithm D inverting f with probability $1 \notin \mathcal{N}$, which contradicts the one-wayness of f.

Construction of D Let $D(y, z) := \bar{\mu}(\bar{L}_f, y, z)$. We have $(*)$ $D(f(x), 1^{|x|}) = \bar{\mu}(\bar{L}_f, f(x), 1^{|x|}) = \mu(L_f, f(x), 1^{|x|})$ by proposition 8.29.

D inverts f Let $x \in \{0,1\}^*$ and let X be as in definition 8.23 with $L := L_f$.

$D(f(x), 1^{|x|})\downarrow$ We have $x2f(x)21^{|x|} \in L_f$ by definition of L_f, so $X \neq \emptyset$ by definition of X, thus $\mu(L_f, f(x), 1^{|x|})\downarrow$ by definition of μ, that is $D(f(x), 1^{|x|})\downarrow$ by $(*)$.

$f\big(D(f(x), 1^{|x|})\big) = f(x)$ We have $D(f(x), 1^{|x|}) = \mu(L_f, f(x), 1^{|x|}) = \min_{\text{lex}} X \in X$ by $(*)$, so $D(f(x), 1^{|x|})2f(x)21^{|x|} \in L_f$ by definition of X, thus $f(D(f(x), 1^{|x|})) = f(x)$ by definition of L_f.

D runs in polynomial time We have that $\bar{L}_f$ is P by hypothesis, so $\bar{\mu}(\bar{L}_f, \cdot, \cdot)$ is polynomial-time computable by inspection of the definition of $\bar{\mu}$, thus D runs in polynomial-time by definition of D.

8.33. We see two advantages in proofs 8.27, 8.30 and 8.32 over proof 8.13:

1. $L_f \rightsquigarrow \bar{L}_f$, $\mu \rightsquigarrow \bar{\mu}$ and $\bar{L}_f$ not being NP $\setminus$ P are not dealt with all at the same time and mixed up but instead separately, namely

   (a) proof 8.27 deals with $L_f \rightsquigarrow \bar{L}_f$;

   (b) proof 8.30 deals with $\mu \rightsquigarrow \bar{\mu}$;

   (c) proof 8.32 deals with $\bar{L}_f$ not being NP $\setminus$ P;

   making it easier to read and check the proofs;

2. the "mini-theory" of minimisation operators μ and $\bar{\mu}$

   (a) is of interest on its own;

   (b) can be reused in other proofs because it was stated explicitly and in the general case in propositions 8.26 and 8.29 instead of implicitly and in a particular case in proof 8.13.

8.5 Conclusion

8.34. In this chapter:

1. we solved the instance of our problem format specified by

$$\begin{aligned} P_1 &= \text{binary-string function,} \\ S_1 &= \text{one-wayness,} \\ P_2 &= \text{formal language,} \\ S_2 &= (\mathrm{NP} \setminus \mathrm{P})\text{-ness,} \\ T &= \text{transformation of a binary-string function} \\ &\quad\; \text{into its induced bitwise formal language;} \end{aligned}$$

2. we gave a proof presentation by "carving out" a "mini-theory" of minimisation operators.

Chapter 9

Transforming cryptographically-secure pseudorandom generators into one-way binary-string functions

9.1 Introduction

9.1. In this chapter we solve the instance of our problem format (presented in paragraph 1.28) specified by the following instances of its parameters:

$$\begin{aligned} P_1 &= \text{pseudorandom generator}, \\ S_1 &= \text{cryptographic security}, \\ P_2 &= \text{binary-string function}, \\ S_2 &= \text{one-wayness}, \\ T &= \text{transformation of a pseudorandom generator} \\ &\quad\ \text{into its induced binary-string function.} \end{aligned}$$

As a proof presentation, instead of proving the security of T by a direct proof as in "if P_1 is S_1-secure, then P_2 is S_2-secure", we see how to prove it by an indirect proof (a proof by contrapositive known as proof by reduction) as in "if P_2 is not S_2-secure (and B breaks the S_2-security of P_2), then P_1 is not S_1-secure (and $A = f(B)$ breaks the S_1-security of P_2)".

As an extra, we consider Oded Goldreich's previously known transformation, prove a relation between our transformation and Oded Goldreich's transformation and present new one-way-preserving transformations in the process.

9.2. Let us informally explain our instances of the parameters.

Pseudorandom generator It is a deterministic algorithm that outputs a stream of bits such as 001101100.

Cryptographic security It means that the stream output by the pseudorandom generator looks random such as 0011011001 in contrast to 0101010101.

Binary-string function It is a function that inputs and outputs binary strings.

One-wayness It means that the function f is easy to compute but hard to invert, that is $x \mapsto f(x)$ is easy to compute but $f(x) \mapsto x$ (or more precisely, $f(x) \mapsto x'$ with $f(x) = f(x')$) is hard to compute.

Transformation It inputs a pseudorandom generator and outputs the binary-string function that on an input x outputs a stream of the pseudorandom generator seeded with x and of length $2|x|$.

9.3. In this chapter all definitions, theorems and proofs are ours except for Oded Goldreich's

1. definition of g_G (Goldreich 2004, proposition 3.3.8);
2. results about g_G in paragraph 9.26 (Goldreich 2004, propositions 2.2.3 and 3.3.8).

9.2 Transformation

9.4. Informally, a binary-string function f_G induced by a pseudorandom generator G is obtained by passing the input x of $f_G(x)$ as a seed to G, getting the stream $G(x, \ldots)$ of G, and taking that stream as the output of f_G, that is $f_G(x) := G(x, \ldots)$. It remains to choose how long should the stream of G be:

1. we are tempted to choose the length $|x|$, that is $f_G(x) := G(x, 1^{|x|})$, but this seemingly does not allow to prove that f_G is one-way from the assumption that G is cryptographically secure because of hard-to-explain technical reasons (a certain probability β in proof 9.10 needs to be negligible);
2. so instead we choose the length $2|x|$, that is $f_G(x) := G(x, 1^{2|x|})$, which solves the problem (by making $\beta \leq 2^{-n}$ negligible).

Now we formally define the binary-string function induced by a pseudorandom generator.

9.5 Definition. The (total) *binary-string function f_G induced by a pseudorandom generator G* is $f_G(x) := G(x, 1^{2|x|})$.

9.6. Now, to be sure, we explicitly state the transformation in question.

9.7 Definition. The *transformation* of a pseudorandom generator G into its induced binary-string function f_G is $G \rightsquigarrow f_G$.

9.3 Security

9.8. Now we show that if the pseudorandom generator G is cryptographically secure, then its induced binary-string function f_G is one-way. Informally, this means that secure (in some sense) pseudorandom generators are transformed into secure (also in some sense) binary-string functions.

9.9 Theorem. For all pseudorandom generators G, if G is cryptographically secure, then f_G is one-way.

9.10 Proof. The pseudorandom generator G being cryptographically secure means

$$\begin{aligned}\forall A, A' \ \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \\ \Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N},\end{aligned} \tag{9.1}$$

where A and A' range over the polynomial-time probabilistic algorithms. The binary-string function f_G being one-way means f_G is polynomial-time computable, which is the case because G, $|\cdot|$, $2\cdot$ and $1^{\cdot}$ are polynomial-time computable, and

$$\forall B \ \Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big] \in \mathcal{N}, \tag{9.2}$$

where B ranges over the polynomial-time probabilistic algorithms.

Taking $A(x) := (1^{2|x|}, \epsilon)$ and $A'(x, y, z) := \mathrm{Tr}[f_G(B(x, y)) = x]$, which are polynomial-time probabilistic algorithms because B is a polynomial-time probabilistic algorithm and $|\cdot|$, $2\cdot$, $1^{\cdot}$, ϵ, $(\cdot, \cdot)$, f_G and $\mathrm{Tr}[\cdot = \cdot]$ are polynomial-time computable, in (9.1), we get

$$\begin{aligned}\forall B \ \Pr \mathrm{Tr}\big[f_G\big(B(G(U_n, 1^{2n}), 1^n)\big) = G(U_n, 1^{2n})\big] - \\ \Pr \mathrm{Tr}[f_G(B(U_{|G(U_n, 1^{2n})|}, 1^n)) = U_{|G(U_n, 1^{2n})|}] \in \mathcal{N}.\end{aligned}$$

Substituting $\Pr \mathrm{Tr}\, P$ by $\Pr P$, where P is any predicate, we get

$$\begin{aligned}\forall B \ \Pr\big[f_G\big(B(G(U_n, 1^{2n}), 1^n)\big) = G(U_n, 1^{2n})\big] - \\ \Pr[f_G(B(U_{|G(U_n, 1^{2n})|}, 1^n)) = U_{|G(U_n, 1^{2n})|}] \in \mathcal{N}.\end{aligned}$$

Substituting $G(U_n, 1^{2n})$ by $f_G(U_n)$, by definition of f_G, we get

$$\begin{aligned}\forall B \ \Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big] - \\ \Pr[f_G(B(U_{|G(U_n, 1^{2n})|}, 1^n)) = U_{|G(U_n, 1^{2n})|}] \in \mathcal{N}.\end{aligned}$$

Substituting $|G(U_n, 1^{2n})|$ by $2n$, by definition of G, we get

$$\forall B \ \underbrace{\Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big]}_{=:\alpha} - \underbrace{\Pr[f_G(B(U_{2n}, 1^n)) = U_{2n}]}_{=:\beta} \in \mathcal{N}.$$

For all probabilistic polynomial-time algorithms B, we have the following.

$\underline{\alpha - \beta \in \mathcal{N}}$ It is stated above.

$\underline{\beta \in \mathcal{N}}$ We have $\forall x \in \{0,1\}^* \, \forall n \in \mathbb{N} \, (x \in \{0,1\}^n \Leftrightarrow f_G(x) \in \{0,1\}^{2n})$ by definition of f_G and G, so $f_G^{-1}[\{0,1\}^{2n}] \subseteq \{0,1\}^n$, thus $0 \leq \beta \leq |f_G^{-1}[\{0,1\}^{2n}]|/|\{0,1\}^{2n}| \leq |\{0,1\}^n|/|\{0,1\}^{2n}| = 2^{-n} \in \mathcal{N}$ because U_{2n} ranges uniformly over $\{0,1\}^{2n}$, hence $\beta \in \mathcal{N}$ because $\mathcal{N}$ is downwards closed in the sense of $\forall f, g \ (0 \leq f \leq g \in \mathcal{N} \Rightarrow f \in \mathcal{N})$ (Gaspar and Boiten 2014, section 2.5), where f and g range over the functions from some set L to $\mathbb{R}$.

$\underline{\alpha - \beta, \beta \in \mathcal{N} \ \Rightarrow \ \alpha \in \mathcal{N}}$ Follows from $\alpha = (\alpha - \beta) + \beta$ and $\mathcal{N}$ being closed under addition in the sense of $\forall f, g \in \mathcal{N}\ f + g \in \mathcal{N}$ (Katz and Lindell 2015, point 1 of proposition 3.6), where f and g range over the functions from some set L to $\mathbb{R}$.

From the three points above we get $\alpha \in \mathcal{N}$, so (9.2).

Schematically, the proof is the following:

$$\forall A, A'\ \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N}$$

$$\Big\Downarrow \ \begin{array}{c} A(x) := (1^{2|x|}, \epsilon) \\ A'(x,y,z) := \mathrm{Tr}[f_G(B(x,y)) = x] \end{array}$$

$$\forall B\ \Pr \mathrm{Tr}\big[f_G\big(B(G(U_n, 1^{2n}), 1^n)\big) = G(U_n, 1^{2n})\big] - \Pr \mathrm{Tr}[f_G(B(U_{|G(U_n,1^{2n})|}, 1^n)) = U_{|G(U_n,1^{2n})|}] \in \mathcal{N}$$

$$\Big\Downarrow \ \Pr \mathrm{Tr}\, P = \Pr P$$

$$\forall B\ \Pr\big[f_G\big(B(G(U_n, 1^{2n}), 1^n)\big) = G(U_n, 1^{2n})\big] - \Pr[f_G(B(U_{|G(U_n,1^{2n})|}, 1^n)) = U_{|G(U_n,1^{2n})|}] \in \mathcal{N}$$

$$\Big\Downarrow \ G(U_n, 1^{2n}) = f_G(U_n)$$

$$\forall B\ \Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big] - \Pr[f_G(B(U_{|G(U_n,1^{2n})|}, 1^n)) = U_{|G(U_n,1^{2n})|}] \in \mathcal{N}$$

$$\Big\Downarrow \ |G(U_n, 1^{2n})| = 2n$$

$$\forall B\ \underbrace{\Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big]}_{=:\alpha} - \underbrace{\Pr[f_G(B(U_{2n}, 1^n)) = U_{2n}]}_{=:\beta} \in \mathcal{N}$$

$$\Big\Downarrow \ \begin{array}{c} \beta \in \mathcal{N} \\ \alpha - \beta, \beta \in \mathcal{N} \ \Rightarrow \ \alpha \in \mathcal{N} \end{array}$$

$$\forall B\ \Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big] \in \mathcal{N}$$

9.4 Proof presentation: indirect proof

9.11. We are going to improve the proof presentation of proof 9.10 (motivated by the reasons given in section 1.5).

9.12 Proof. The pseudorandom generator G *not* being cryptographically secure means

$$\exists A, A'\ \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \notin \mathcal{N}, \tag{9.3}$$

where A and A' range over the polynomial-time probabilistic algorithms. The binary-string function f_G *not* being one-way means that f_G is not polynomial-time computable, which is not the case because G, $|\cdot|$, $2\cdot$ and $1^{\cdot}$ are polynomial-time computable, or

$$\exists B\ \Pr\big[f_G\big(B(f_G(U_n), 1^n)\big) = f_G(U_n)\big] \notin \mathcal{N}, \tag{9.4}$$

where B ranges over the polynomial-time probabilistic algorithms.

Let

$$\begin{aligned}
\alpha &:= \Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr],\\
\beta &:= \Pr[f_G(B(U_{2n}, 1^n)) = U_{2n}].
\end{aligned}$$

First, let us prove $\beta \in \mathcal{N}$ and $\alpha - \beta, \beta \in \mathcal{N} \ \Rightarrow\ \alpha \in \mathcal{N}$.

$\underline{\beta \in \mathcal{N}}$ We have $\forall x \in \{0,1\}^* \ \forall n \in \mathbb{N}\ (x \in \{0,1\}^n \ \Leftrightarrow\ f_G(x) \in \{0,1\}^{2n})$ by definition of f_G and G, so $f_G^{-1}[\{0,1\}^{2n}] \subseteq \{0,1\}^n$, thus $0 \leq \beta \leq |f_G^{-1}[\{0,1\}^{2n}]|/|\{0,1\}^{2n}| \leq |\{0,1\}^n|/|\{0,1\}^{2n}| = 2^{-n} \in \mathcal{N}$ because U_{2n} ranges uniformly over $\{0,1\}^{2n}$, hence $\beta \in \mathcal{N}$ because $\mathcal{N}$ is downwards closed in the sense of $\forall f, g\ (0 \leq f \leq g \in \mathcal{N} \ \Rightarrow\ f \in \mathcal{N})$ (Gaspar and Boiten 2014, section 2.5), where f and g range over the functions from some set L to $\mathbb{R}$.

$\underline{\alpha - \beta, \beta \in \mathcal{N} \ \Rightarrow\ \alpha \in \mathcal{N}}$ Follows from $\alpha = (\alpha - \beta) + \beta$ and $\mathcal{N}$ being closed under addition in the sense of $\forall f, g \in \mathcal{N}\ f + g \in \mathcal{N}$ (Katz and Lindell 2015, point 1 of proposition 3.6), where f and g range over the functions from some set L to $\mathbb{R}$.

Second, let us schematically prove (9.3) from (9.4).

$$\begin{gathered}
\exists A, A'\ \ \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - {}\\
\Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \notin \mathcal{N}
\end{gathered}$$

$$\Big\Uparrow\ \begin{array}{l} A(x) := (1^{2|x|}, \epsilon)\\ A'(x, y, z) := \mathrm{Tr}[f_G(B(x, y)) = x] \end{array}$$

$$\begin{gathered}
\exists B\ \ \Pr\mathrm{Tr}\bigl[f_G\bigl(B(G(U_n, 1^{2n}), 1^n)\bigr) = G(U_n, 1^{2n})\bigr] - {}\\
\Pr\mathrm{Tr}[f_G(B(U_{|G(U_n, 1^{2n})|}, 1^n)) = U_{|G(U_n, 1^{2n})|}] \notin \mathcal{N}
\end{gathered}$$

$$\Big\Uparrow\ \Pr P = \Pr\mathrm{Tr}\,P$$

$$\begin{gathered}
\exists B\ \Pr\bigl[f_G\bigl(B(G(U_n, 1^{2n}), 1^n)\bigr) = G(U_n, 1^{2n})\bigr] - {}\\
\Pr[f_G(B(U_{|G(U_n, 1^{2n})|}, 1^n)) = U_{|G(U_n, 1^{2n})|}] \notin \mathcal{N}
\end{gathered}$$

$$\Big\Uparrow\ f_G(U_n) = G(U_n, 1^{2n})$$

$$\begin{gathered}
\exists B\ \Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr] - {}\\
\Pr[f_G(B(U_{|G(U_n, 1^{2n})|}, 1^n)) = U_{|G(U_n, 1^{2n})|}] \notin \mathcal{N}
\end{gathered}$$

$$\Big\Uparrow\ 2n = |G(U_n, 1^{2n})|$$

$$\exists B\ \ \underbrace{\Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr]}_{=\alpha} - \underbrace{\Pr[f_G(B(U_{2n}, 1^n)) = U_{2n}]}_{=\beta} \notin \mathcal{N}$$

$$\Big\Uparrow\ \begin{array}{c} \beta \in \mathcal{N}\\ \alpha - \beta, \beta \in \mathcal{N} \ \Rightarrow\ \alpha \in \mathcal{N} \end{array}$$

$$\exists B\ \ \underbrace{\Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr]}_{=\alpha} \notin \mathcal{N}$$

9.13. We see an advantage and a disadvantage in proof 9.12 over proof 9.10:

1. the advantage is that most people find it more natural to

   > transform an algorithm B breaking the one-wayness of f_G (the B in $\exists B\ \neg \ldots$) into an algorithm A breaking the cryptographic security of G (the A in $\exists A\ \neg \ldots$)

   than to

   > transform the statement that all algorithms A do not break the cryptographic security of G (the statement $\forall A\ \ldots$) into the statement that all algorithms B do not break the one-wayness of f_G (the statement $\forall B\ \ldots$).

2. the disadvantage is that most people find it more elegant a direct proof (as in $\forall A\ \ldots\ \Rightarrow\ \forall B\ \ldots$) than an indirect proof (as in $\exists B\ \neg \ldots\ \Rightarrow\ \exists A\ \neg \ldots$).

9.5 Extra: Oded Goldreich's transformation

9.14 Definition. Our new *total* binary-string function $f_G(x) := G(x, 1^{2|x|})$, where $x \in \{0,1\}^*$, is similar to Oded Goldreich's old *partial* binary-string function $g_G(x) := G(x_{\leftarrow}, 1^{|x|})$, where $x \in \{0,1\}^{2*}$ is only allowed to have even lengths. Now we study the relation between f_G and g_G and conclude that they are the same modulo certain one-wayness-preserving operators $\cdot_\mathrm{p}$ and $\cdot_\mathrm{t}$ that transform total binary-string functions into partial binary-string functions and vice-versa, in the sense of $(f_G)_\mathrm{p} = g_G$ and $(g_G)_\mathrm{t} = f_G$.

Let us start by introducing g_G.

9.15 Definition. The *partial binary-string function* $g_G\colon \{0,1\}^{2*} \to \{0,1\}^*$ *induced by a pseudorandom generator* G (Goldreich 2004, proposition 3.3.8) is $g_G(x) := G(x_{\leftarrow}, 1^{|x|})$.

9.16. Now let us introduce the operators $\cdot_\mathrm{p}$ and $\cdot_\mathrm{t}$ and prove that they preserve one-wayness.

9.17 Definition.

1. Given a total binary-string function $f\colon \{0,1\}^* \to \{0,1\}^*$, we define the partial binary-string function $f_\mathrm{p}\colon \{0,1\}^{2*} \to \{0,1\}^*$ by $f_\mathrm{p}(x) := f(x_{\leftarrow})$.

2. Given a partial binary-string function $g\colon \{0,1\}^{2*} \to \{0,1\}^*$, we define the total binary-string function $g_\mathrm{t}\colon \{0,1\}^* \to \{0,1\}^*$ by $g_\mathrm{t}(x) := g(x1^{|x|})$.

9.18 Remark. The partial binary-string function f_p does not depend on the right half of the input.

9.19 Proposition. For all binary-string functions $f\colon \{0,1\}^* \to \{0,1\}^*$, and for all partial binary-string functions $g\colon \{0,1\}^{2*} \to \{0,1\}^*$ that do not depend on the right half the input, we have:

1. if f is one-way, then f_{p} is one-way;

2. if g is one-way, then g_{t} is one-way.

9.20 Proof.

1. The total binary-string function f being one-way means that f is polynomial-time computable and

$$\forall A \ \Pr\bigl[f\bigl(A(f(U_n), 1^n)\bigr) = f(U_n)\bigr] \in \mathcal{N}, \tag{9.5}$$

where A ranges over the polynomial-time probabilistic algorithms. The partial binary-string function f_{p} being one-way means that f_{p} is polynomial-time computable, which is the case because f and $\cdot_{\leftarrow}$ are polynomial-time computable, and

$$\forall B \ \underbrace{\Pr\bigl[f_{\mathrm{p}}\bigl(B(f_{\mathrm{p}}(U_n), 1^n)\bigr) = f_{\mathrm{p}}(U_n)\!\downarrow\bigr]}_{:=\alpha_n} \in \mathcal{N}, \tag{9.6}$$

where B ranges over the polynomial-time probabilistic algorithms.

Taking $A(x, y) := B(x, 1^{2|y|})_{\leftarrow}$, which is a polynomial-time probabilistic algorithm because B is a polynomial-time probabilistic algorithm and $|\cdot|$, $2\cdot$, $1^{\cdot}$ and $\cdot_{\leftarrow}$ are polynomial-time computable, in (9.5), we get

$$\forall B \ \Pr\bigl[f\bigl(B(f(U_n), 1^{2n})_{\leftarrow}\bigr) = f(U_n)\bigr] \in \mathcal{N}.$$

Using $\Pr\bigl[f_{\mathrm{p}}\bigl(B(f(U_n), 1^{2n})\bigr) = f(U_n)\!\downarrow\bigr] = \Pr\bigl[f\bigl(B(f(U_n), 1^{2n})_{\leftarrow}\bigr) = f(U_n) \wedge B(f(U_n), 1^{2n}) \in \{0,1\}^{2*}\bigr] \leq \Pr\bigl[f\bigl(B(f(U_n), 1^{2n})_{\leftarrow}\bigr) = f(U_n)\bigr]$, by definition of f_{p} and $\mathrm{dom}(f_{\mathrm{p}}) = \{0,1\}^{2*}$, and using that $\mathcal{N}$ is downwards closed, we get

$$\forall B \ \Pr\bigl[f_{\mathrm{p}}\bigl(B(f(U_n), 1^{2n})\bigr) = f(U_n)\!\downarrow\bigr] \in \mathcal{N}.$$

Substituting U_n by $U_{2n\leftarrow}$, because both are uniform random variables in $\{0,1\}^n$, we get

$$\forall B \ \Pr\bigl[f_{\mathrm{p}}\bigl(B(f(U_{2n\leftarrow}), 1^{2n})\bigr) = f(U_{2n\leftarrow})\!\downarrow\bigr] \in \mathcal{N}.$$

Substituting $f(U_{2n\leftarrow})$ by $f_{\mathrm{p}}(U_{2n})$, by definition of f_{p} and $U_{2n} \in \{0,1\}^{2*} = \mathrm{dom}(f_{\mathrm{p}})$, we get

$$\forall B \ \underbrace{\Pr\bigl[f_{\mathrm{p}}\bigl(B(f_{\mathrm{p}}(U_{2n}), 1^{2n})\bigr) = f_{\mathrm{p}}(U_{2n})\!\downarrow\bigr]}_{=\alpha_{2n}} \in \mathcal{N}.$$

Using that

(a) i. $\alpha_n \in \mathcal{N}$ when α_n is restricted to even ns by the formula above;
    ii. $\alpha_n = 0$ when α_n is restricted to odd ns because then $f_{\mathrm{p}}(U_n)\!\uparrow$;

(b) $\mathcal{N}$ is closed under extensions by zeros in the sense that if $h\colon L \subseteq \mathbb{N} \to \mathbb{R}$ is in $\mathcal{N}$, then $h'\colon \mathbb{N} \to \mathbb{R}$ defined by $h'(x) := \begin{cases} h(x) & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$ is in $\mathcal{N}$;

we get (9.6).

2. The partial binary-string function g being one-way means that g is polynomial-time computable and

$$\forall A \ \Pr\big[g\big(A(g(U_n), 1^n)\big) = g(U_n)\downarrow\big] \in \mathcal{N}, \tag{9.7}$$

where A ranges over the polynomial-time probabilistic algorithms. The total binary-string function g_t being one-way means that g_t is polynomial-time computable, which is the case because f, $|\cdot|$, $1^\cdot$ and string concatenation are polynomial-time computable, and

$$\forall B \ \Pr\big[g_\mathrm{t}\big(B(g_\mathrm{t}(U_n), 1^n)\big) = g_\mathrm{t}(U_n)\big] \in \mathcal{N}, \tag{9.8}$$

where B ranges over the polynomial-time probabilistic algorithms.

From (9.7), $\mathrm{dom}(g) = \{0,1\}^{2*}$ and $\mathcal{N}$ being closed under speed-up in the sense $\forall f(n) \in \mathcal{N}\ f(2n) \in \mathcal{N}$, we get

$$\forall A \ \Pr\big[g\big(A(g(U_{2n}), 1^{2n})\big) = g(U_{2n})\big] \in \mathcal{N}.$$

Taking $A(x,y) := B(x, 1^{\lfloor |y|/2 \rfloor})1^{|B(x,1^{\lfloor |y|/2 \rfloor})|}$, which is a polynomial-time probabilistic algorithm because B is a polynomial-time probabilistic algorithm and $|\cdot|$, $\lfloor \cdot/2 \rfloor$ and $1^\cdot$ are polynomial-time computable, we get

$$\forall B \ \Pr\big[g\big(B(g(U_{2n}), 1^n)1^{|B(g(U_{2n}),1^n)|}\big) = g(U_{2n})\big] \in \mathcal{N}.$$

Substituting $g\big(B(g(U_{2n}), 1^n)1^{|B(g(U_{2n}),1^n)|}\big)$ by $g_\mathrm{t}(B(g(U_{2n}), 1^n))$, by definition of g_t, we get

$$\forall B \ \Pr\big[g_\mathrm{t}\big(B(g(U_{2n}), 1^n)\big) = g(U_{2n})\big] \in \mathcal{N}.$$

Substituting $g(U_{2n})$ by $g(U_{2n\leftarrow}1^n)$, because g does not depend on the right half of the input, we get

$$\forall B \ \Pr\big[g_\mathrm{t}(B(g(U_{2n\leftarrow}1^n), 1^n)\big) = g(U_{2n\leftarrow}1^n)\big] \in \mathcal{N}.$$

Substituting $g(U_{2n\leftarrow}1^n)$ by $g_\mathrm{t}(U_{2n\leftarrow})$, by definition of g_t, we get

$$\forall B \ \Pr\big[g_\mathrm{t}\big(B(g_\mathrm{t}(U_{2n\leftarrow}), 1^n)\big) = g_\mathrm{t}(U_{2n\leftarrow})\big] \in \mathcal{N}.$$

Substituting $U_{2n\leftarrow}$ by U_n, because both are uniform random variables in $\{0,1\}^n$, we get (9.8).

9.21 Remark. Point 2 may be false (assuming the existence of one-way functions) if g depends on the right half of the input (proof sketch: if $f\colon \{0,1\}^* \to \{0,1\}^*$ is one-way, then $g\colon \{0,1\}^{2*} \to \{0,1\}^*$ defined by $g(x) := f(x_\rightarrow)$ is one-way analogously to point 1 of proposition 9.19, so $g_\mathrm{t}(x) = g(x1^{|x|}) = f((x1^{|x|})_\rightarrow) = f(1^{|x|})$ is constant over each $\{0,1\}^n$, thus it is not one-way).

9.22. Let us momentarily:

1. denote $\{0,1\}$ by 2 (as done with the von Neumann ordinals in set theory);

2. denote the set of functions from X to Y by $X \to Y$ (which below will read better than the more usual Y^X);
3. extend the definition of f_G and g_G to arbitrary $G \in 2^* \times 2^* \to 2^*$ (instead of only pseudorandom generators G).

The next proposition says (modulo the extension) that the following diagram commutes.

$$
\begin{tikzcd}
 & 2^*\to 2^* \arrow[dd, shift left, "\cdot_{\mathrm{p}}"] \\
2^*\times 2^*\to 2^* \arrow[ur, "f."] \arrow[dr, "g."'] & \\
 & 2^{2*}\to 2^* \arrow[uu, shift left, "\cdot_{\mathrm{t}}"]
\end{tikzcd}
$$

So we can "meta-transform" between our transformation

$$G \text{ cryptographically-secure } \rightsquigarrow f_G \text{ one-way}, \qquad \text{where } f_G = (g_G)_{\mathrm{t}}$$

and Oded Goldreich's transformation

$$G \text{ cryptographically-secure } \rightsquigarrow g_G \text{ one-way}, \qquad \text{where } g_G = (f_G)_{\mathrm{p}}.$$

9.23 Proposition. For all pseudorandom generators G, we have:

1. $(f_G)_{\mathrm{p}} = g_G$;
2. $(g_G)_{\mathrm{t}} = f_G$.

9.24 Proof.

1. We have $\mathrm{dom}((f_G)_{\mathrm{p}}) = \{0,1\}^{2*} = \mathrm{dom}(g_G)$ and
$$\forall x \in \{0,1\}^{2*}\ (f_G)_{\mathrm{p}}(x) = f_G(x_{\leftarrow}) = G(x_{\leftarrow}, 1^{2|x_{\leftarrow}|}) = G(x_{\leftarrow}, 1^{|x|}) = g_G(x).$$
2. We have $\mathrm{dom}((g_G)_{\mathrm{t}}) = \{0,1\}^* = \mathrm{dom}(f_G)$ and
$$\forall x \in \{0,1\}^*\ (g_G)_{\mathrm{t}}(x) = g_G(x1^{|x|}) = G((x1^{|x|})_{\leftarrow}, 1^{|x1^{|x|}|}) = G(x, 1^{2|x|}) = f_G(x).$$

9.25. We give to proposition 9.19 two interpretations (only) seemingly contradictory:

1. on the one hand, the proposition says that f_G and g_G are (modulo $\cdot_{\mathrm{p}}$ and $\cdot_{\mathrm{t}}$) the same function, so in this way the two constructions $f.$ of $g.$ of one-way binary-string functions from cryptographically-secure pseudorandom generators are the same;
2. on the other hand, to infer the one-wayness of f_G from g_G and vice-versa, it takes (the about two pages of)
   (a) definition 9.17;

(b) proposition 9.19;

(c) proof 9.20;

so in this way the two constructions $f.$ of $g.$ are (non-trivially) separated.

9.26. There are two properties that are desired in a one-way partial binary-string f: to be extensible to

1. a *length-preserving* one-way partial binary-string;

2. a *total* one-way binary-string;

or even

3. a *length-preserving* and *total* one-way binary-string f'.

In the next table we summarise the existence of these extensions for f_G and g_G:

	length preserving	total	extensible to length preserving and total
f_G	no	yes	no
g_G	yes	no	yes

(proof sketch: f_G is not extensible to a length-preserving function because f_G doubles lengths; the results for g_G are known (Goldreich 2004, propositions 2.2.3 and 3.3.8)). Let us compare f_G and g_G:

1. f_G is better in the sense that the proof that f_G is one-way (if G is cryptographically secure) is simpler by avoiding the delicateness of dealing with partial functions such as g_G;

2. g_G is better in the sense that g_G can be extended to a length-preserving total function while f_G cannot.

9.6 Conclusion

9.27. In this chapter:

1. we solved the instance of our problem format specified by:

$$\begin{aligned} P_1 &= \text{pseudorandom generator}, \\ S_1 &= \text{cryptographic security}, \\ P_2 &= \text{binary-string function}, \\ S_2 &= \text{one-wayness}, \\ T &= \text{transformation of a pseudorandom} \\ &\ \text{generator into a binary-string function}; \end{aligned}$$

2. we gave a proof presentation by means of an indirect proof;

3. we compared our transformation with Oded Goldreich's transformation and presented new one-way-preserving transformations in the process.

Part V

Conclusion

Chapter 10

Provable security of transformation of cryptographic primitives

10.1 Introduction

10.1. In section 1.2 we identified a problem in cryptography: definitions, theorems and proofs in cryptography are often so complicated that some incorrect "proofs" go unnoticed. In sections 1.3, 1.4, 1.5 and 1.6 we proposed a solution for the problem. The solution is divided into components and two of the components are the transformation of cryptographic primitives and provable security. In section 1.3 we argued that the transformation of cryptographic primitive helps solve the problem by producing simpler security proofs, allowing abstraction from underlying theories and operating on higher level concepts. In section 1.4 we argued that provable security also helps solve the problem by giving a mathematical guarantee of correctness, being amenable to formal verification and helping find mistakes. So, ultimately, in the "big picture" of things, we hope that our work devoted to the provable security of transformation of cryptographic primitives will contribute to the solution of the problem.

10.2. To be more concrete, we give three examples of small insights that we developed.

1. Cryptographic primitives and security notions:
   (a) often require some algorithm $A(x)$ to run in polynomial time in a security parameter n or in the length of the input x;
   (b) sometimes transformations of cryptographic primitives and their security proofs overlook this requirement;
   (c) often this can be spotted by checking if the proof addresses runtimes or what happens when $A(x)$ "shrinks too much" or "expands too much" x;
   (d) sometimes the proof can be fixed by adding 1^n or $1^{|x|}$ as an extra input to A, or adding extra assumptions about lengths, and changing the proof accordingly.

2. Security notions often have variants that differ by

   (a) allowing extra inputs or not;

   (b) requiring (a family of) algorithms to be uniform or not;

   (c) requiring functions to be total or not;

   and so on, and this raises three problems that we need to be aware of:

   (a) when citing a result, we should check if the result's security notions are our ones or if the result's proof can be easily adapted to our ones;

   (b) some security proofs depend in an essential way on the variants of the security notions used so it is dangerous to overlook the question of variants;

   (c) security notions mentioned in the security of transformation of cryptographic need to be compatible variants, often in the sense of having matching allowances and requirements.

3. It is possible for

   (a) a security notion to "make perfect sense";

   (b) a security result to look "obviously true";

   (c) a security proof to be "plainly straightforward";

   and yet

   (a) the security notion being

      Uninstantiable no cryptographic primitive satisfies the security notion;

      Unknown to be (non-trivially) instantiable we do not know how to construct a (non-trivial) cryptographic primitive satisfying the security notion even under common assumptions such as the existence of one-way binary-string functions;

      Instantiable but uninteresting only satisfied by extremely strong cryptographic primitives such as the one-time pad, or turning out to be equivalent to an elementary notion such as injectiveness;

      Universally instantiable being satisfied by every cryptographic primitive, even by the identity function and constant functions.

   (b) the security result missing some minor assumption to be really true;

   (c) the security proof having a "bug" at an "innocent-looking" point;

   so one cannot check enough security notions, security results and their security proofs.

10.3. In the previous chapters we studied the transformations of cryptographic primitives shown in the (directed) graph

$$
\begin{tikzcd}
\begin{array}{c}\text{one-way}\\ \text{binary-string functions}\end{array} \arrow[loop above, "\text{chapter 6}"] \arrow[d, "\text{chapter 8}"'] & \begin{array}{c}\text{cryptographically-secure}\\ \text{pseudorandom generators}\end{array} \arrow[l, "\text{chapter 9}"'] \arrow[d, "\text{chapter 7}"] \\
\begin{array}{c}\text{NP} \setminus \text{P}\\ \text{formal languages}\end{array} & \begin{array}{c}\text{indistinguishable-from-random}\\ \text{stream ciphers}\end{array}
\end{tikzcd}
$$

where

1. each vertex of the graph is a pair (P, S) consisting of

   (a) a cryptographic primitive P;

   (b) a security notion S (for P);

   (if we are interested only in the cryptographic primitive P, then we refer to the vertex (P, S) simply as P; if we are only interested in the security notion S, then we refer to the vertex (P, S) simply as S);

2. each edge of the graph from a vertex (P_1, S_1) to a vertex (P_2, S_2) is a pair (T, P) consisting of

   (a) a transformation T of cryptographic primitives P_1 into cryptographic primitives P_2;

   (b) a security proof P that T preserves security in the sense that if P_1 is S_1-secure, then P_2 (transformed from P_1 by T) is S_2-secure;

   (if we are interested only in the transformation T, then we refer to the edge (T, P) simply as T; if we are only interested in the security proof P, then we refer to the vertex (T, P) simply as P);

(our graph bears some resemblance to a graph with security notions for asymmetric ciphers as vertices, and implications and non-implications as edges (Bellare, Desai, Pointcheval and Rogaway 1998, figure 1)). In this chapter we propose to study more transformations guided through some research lines:

1. completion of the graph of the already seen transformations;

2. addition of new vertices to the graph by varying the security notions or adding new cryptographic primitives;

3. basing the construction of cryptographic primitives on indistinguishable-from-random stream ciphers instead of one-way binary-string functions;

4. extraction of numeric/computational content in line with concrete security;

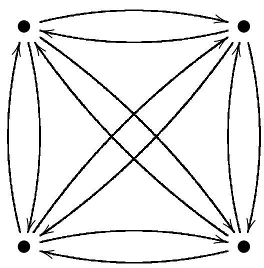
5. composition of transformations into new composed transformation and decomposition of transformations into new decomposing transformations;
6. empirically testing implemented candidates to secure cryptographic primitives obtained by transformations of well-established implemented candidates to secure cryptographic primitives;
7. improving transformation claims or proofs;
8. transformation of cryptographic protocols;
9. recasting cryptographic concepts as topological concepts.

10.4. In this chapter all work is ours except section 10.10, which builds on ideas by Stefan Kahrs and Eerke Boiten.

10.2 Proposal: graph completion

10.5 Proposal.

Idea We propose to complete the graph from section 10.1 into a graph such as the following one.



Comments Of the transformations necessary to complete the graph:

1. some are already known, for example the transformation of one-way binary-string functions into cryptographically-secure pseudorandom generators (Goldreich 2004, theorem 3.5.12);
2. other transformations may be easy, for example the transformation of an indistinguishable-from-random stream cipher into a cryptographically-secure pseudorandom generator may be as easy as to encrypt the plaintext 0^n;
3. other transformations seem unlikely, for example a (non-trivial) transformation of a (worst-case) NP \ P formal language into an (average-case) one-way binary-string function (but there is likely a trivial transformation, namely a constant transformation $L \rightsquigarrow f$, where f is a fixed one-way binary-string function, since likely exists such an f).

Benefit We see the following benefits of this proposal:

1. it gives a "full" description of the transformations between the considered cryptographic primitives;

2. it shows a tight connection between the considered cryptographic primitives by proving that they all stand together in the sense that their existences rise or fall together;
3. even if the transformation (or its proof) from a cryptographic primitive P_1 into a cryptographic primitive P_2 turns out to be wrong, it still holds that P_1 can be transformed into P_2 because there are several other paths from P_1 to P_2;
4. it gives freedom to chose how to transform a cryptographic primitive P_1 into a cryptographic primitive P_2 because there are several paths from P_1 to P_2 (although the more efficient construction is likely the edge from P_1 to P_2).

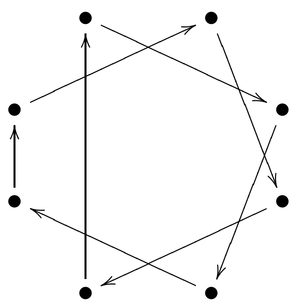
10.3 Proposal: vertices addition

10.6 Proposal.

Idea We proposed to add new vertices to the graph by varying the notions of security or by adding new cryptographic primitives, for example:

1. trapdoor functions;
2. block ciphers;
3. asymmetric ciphers;
4. digital-signature schemes;
5. hash functions;
6. message authentication codes (MAC);
7. commitment schemes.

Comments The cost of this proposal is that if we still aim for a complete graph, then the graph becomes unmanageably large, so we may instead aim for a direct cycle covering all vertices in the graph like the one below because this guarantees that every cryptographic primitive can be transformed into any other cryptographic primitive.



Benefit We see the following benefits of this proposal:

1. the more security notions or cryptographic primitives that are considered, the more complete is the picture;

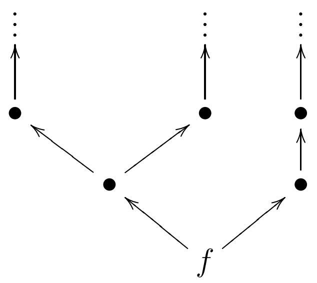
2. if P_1 is an old cryptographic primitive that is conjectured to exist and is already in the graph, and P_2 is a new cryptographic primitive that is added to the graph, then the conjecture on the existence of P_1 implies a conjecture on the existence of P_2;

3. if S_1 is an old security notion that is already in the graph, and S_2 is a new security notion that is added to the graph, then the existence of a path from S_1 to S_2 and vice-versa shows that S_1 and S_2 are equivalent, which is evidence that they are "right" notions since it would be an unlikely coincidence for two "wrong" notions to be equivalent.

10.4 Proposal: basing on indistinguishable-from-random stream ciphers

10.7 Proposal.

Idea We propose to build the array of cryptographic primitives starting with indistinguishable-from-random stream ciphers (or cryptographically-secure pseudorandom generators). The traditional construction starts with one-way binary-string functions in the base, indicated by f in the tree below, and constructs the other cryptographic primitives, indicated by $\bullet$, from one-way binary-string functions (and other primitives constructed in the meantime), as in the following tree.



In our proposal, f in the base would be replaced by indistinguishable-from-random stream ciphers C.

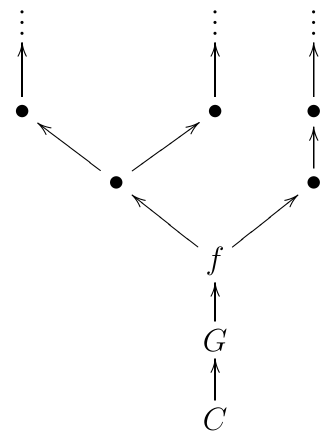
Comments There is one easy way to realise this proposal: to prepend to the above tree the transformations $C \rightsquigarrow G \rightsquigarrow f$

1. starting at indistinguishable-from-random stream ciphers C;

2. proceeding into cryptographically-secure pseudorandom generators G;

3. finishing at one-way binary-string functions f;

obtaining



but this construction is of little interest because it is unlikely to be "optimal" (for example if by optimal we mean with the least possible number of edges, but we can have other meanings in mind such as the best "quality" discussed in section 10.5).

Benefit We see the following benefit of this proposal: it appears to be easier

> to construct the transformations $G \rightsquigarrow P$ of cryptographically-secure pseudorandom generators G into other cryptographic primitives P
>
> (for example we saw the not-too-complicated transformations $G \rightsquigarrow C$ and $G \rightsquigarrow f$ of cryptographically-secure pseudorandom generators G into indistinguishable-from-random stream ciphers C in chapter 7 and into one-way binary-string functions f in chapter 9)

than

> to construct the transformations $f \rightsquigarrow P$ of one-way binary-string functions f into other cryptographic primitives P
>
> (for example Oded Goldreich's book states the transformation $f \rightsquigarrow G$ of one-way binary-string functions f into cryptographically-secure pseudorandom generators G (Goldreich 2004, theorem 3.5.12) but prefers to prove simpler particular cases where f has extra properties instead of the full general case because of its too-complicated proof (Goldreich 2004, section 3.5)).

10.5 Proposal: extraction of numeric/computational content

10.8 Proposal.

Idea We propose to extract numeric/computational content from security proofs of transformations of cryptographic primitives. This is known in cryptography as concrete security (Bellare, Desai, Jokipii and Rogaway 1997) and in logic as proof mining (Kohlenbach 2008). The extraction consists in analysing a known proof and quantifying (or at least finding an upperbound on) a measure of the "quality" of a cryptographic primitive.

Comments For example:

1. instead of saying that a probability $\Pr P(n)$ is negligible, we could measure how fast the probability vanishes as in $\Pr P(n) = 2^{-(n-1)}/3$ (or at least find an upperbound as in $\Pr P(n) \leq 2^{-n}$);
2. instead of saying that an algorithm $A(x)$ is polynomial-time computable, we could measure how fast the algorithm runs as in "$A(x)$ runs exactly in time $|x|^2 + (-1)^{|x|} + 1$" (or at least find an upperbound as in "$A(x)$ runs in time at most $|x|^2 + 2$").

In the same style, we could bring the construction of algorithms breaking the security of cryptographic primitives to the foreground:

3. instead of saying that if there is an algorithm A breaking the security of a cryptographic primitive, then there is an algorithm C breaking the security of another cryptographic primitive, we could present the construction of C from B as in $C(x, y) := B(1^{|x|}, x \oplus y)$ and point out the material $|\cdot|$, $1^{\cdot}$ and $\oplus$ used in this construction.

$$\left.\begin{array}{r} \Pr P(n) \in \mathcal{N} \\ A(x) \in \mathrm{PPT} \\ B(x, y) \rightsquigarrow C(x, y) \end{array}\right\} \quad \longrightarrow \quad \left\{\begin{array}{l} \Pr P(n) = 2^{-(n-1)}/3 \\ \mathrm{Time}(A(x)) = |x|^2 + (-1)^{|x|} + 1 \\ C(x, y) := B(1^{|x|}, x \oplus y) \\ |\cdot|,\ 1^{\cdot},\ \oplus \end{array}\right.$$

Benefit We see the following benefits of this proposal:

1. having numeric/computational information on cryptographic primitives may allow to estimate the size of security parameters to achieve a desired level of security (Koblitz and Menezes 2007, pages 17 and 29) (Koblitz and Menezes 2006, sections 4 and 6);
2. the numeric/computational information may help to break a tie between two competing cryptographic primitives or transformations;
3. the numeric/computational information may help avoiding less interesting transformations such as transforming a polynomial-time computable cryptographic primitive into another cryptographic primitive by means of an exponential-time computable transformation;
4. the numeric/computational content may indicate the power of the hardware necessary to implement a transformed cryptographic primitive, which may be relevant when the cryptographic primitive is intended to run in low-end hardware or very fast.

10.6 Proposal: composition and decomposition

10.9 Proposal.

Idea We propose to improve our understanding of already-known transformations of cryptographic primitives by means of composition or, inversely, decomposition.

Composition Given the transformations $P_1 \overset{T_1}{\rightsquigarrow} P_2$ and $P_2 \overset{T_2}{\rightsquigarrow} P_3$, we can produce the composed transformation $P_1 \overset{T_2 \circ T_1}{\rightsquigarrow} P_3$.

Decomposition Given the transformation $P_1 \overset{T}{\rightsquigarrow} P_3$, maybe we can produce decomposing transformations $P_1 \overset{T_1}{\rightsquigarrow} P_2$ and $P_2 \overset{T_2}{\rightsquigarrow} P_3$ such that $T = T_2 \circ T_1$.

$$\begin{tikzcd} P_1 \arrow[r, "T_1"] \arrow[rr, bend right, "T = T_2 \circ T_1"'] & P_2 \arrow[r, "T_2"] & P_3 \end{tikzcd}$$

Comments Let us see an example of a composition and an example of a decomposition.

Composition In chapter 9 we saw the composition of $g_\cdot$ and $\cdot_{\mathrm{t}}$, which gives $f_\cdot$;

Decomposition In chapter 9 we saw the decomposition of $f_\cdot$ into $g_\cdot$ and $\cdot_{\mathrm{t}}$.

$$\begin{tikzcd} G \arrow[r, "g_\cdot"] \arrow[rr, bend right, "f_\cdot = \cdot_{\mathrm{t}} \circ g_\cdot"'] & g_G \arrow[r, "\cdot_{\mathrm{t}}"] & (g_G)_{\mathrm{t}} = f_G \end{tikzcd}$$

Benefit We see the following benefits of this proposal.

Composition If the old transformations $P_1 \overset{T_1}{\rightsquigarrow} P_2$ and $P_2 \overset{T_2}{\rightsquigarrow} P_3$ are composed into a new transformation $P_1 \overset{T_2 \circ T_1}{\rightsquigarrow} P_3$, then:

1. it is trivial to get a security proof of $T_2 \circ T_1$ by composing the security proofs for T_1 and T_2 but we may aim at a simpler security proof for $T_2 \circ T_1$ using the fact that T_2 in $T_2 \circ T_1$ is not applied to arbitrary cryptographic primitives P_2 but only to cryptographic primitives P_2 transformed from P_1 by T_1;
2. it may be possible to simplify $T_2 \circ T_1$ into a new transformation T and so $T_2 \circ T_1$ guided the discovery of T.

Decomposition If the old transformation $P_1 \overset{T}{\rightsquigarrow} P_3$ is decomposed into new transformations $P_1 \overset{T_1}{\rightsquigarrow} P_2$ and $P_2 \overset{T_2}{\rightsquigarrow} P_3$, then:

1. the decomposition may reveal new concepts underlying P_2 and its security notion;
2. the security proofs for $P_1 \overset{T_1}{\rightsquigarrow} P_2$ and $P_2 \overset{T_2}{\rightsquigarrow} P_3$ may be simpler than the security proof for $P_1 \overset{T}{\rightsquigarrow} P_3$ because, say,

(a) the security proof for T_1 only uses probability theory;
(b) the security proof for T_2 only uses computation theory;
(c) the security proof of T mixes both theories.

10.7 Proposal: empirical tests

10.10 Proposal.

Idea We propose to empirically test a transformation $P_1 \overset{T}{\rightsquigarrow} P_2$ from an S_1-secure cryptographic primitive P_1 to an S_2-secure cryptographic primitive P_2 by:

1. instantiating P_1 by a well-established implemented candidate to an S_1-secure cryptographic primitive P_1;
2. applying the transformation T to P_1 to obtain an implemented candidate to an S_2-secure cryptographic primitive P_2;
3. test the candidate to P_2 with a test suite S.

$$P_1 \xrightarrow{\text{transform with } T} P_2 \xrightarrow{\text{test with}} S$$

Comments For example, if the transformation T consists in transforming a block cipher C into a pseudorandom generator G by running the block cipher in the counter mode of operation, then we could think of:

1. instantiate C with the Advanced Encryption Standard (AES), which is a well-established candidate to a secure block cipher;
2. applying the transformation T to C to obtain an implemented candidate to a secure pseudorandom generator G;
3. test G with the Diehard tests (a popular battery of statistical tests of randomness).

$$\text{AES} \xrightarrow{\substack{\text{transform with}\\ \text{counter mode}}} \substack{\text{pseudorandom}\\ \text{generator}} \xrightarrow{\text{test with}} \text{Diehard}$$

Benefit We see the following benefits of this proposal:

1. it opens the door for applied empirical tests of transformations of cryptographic primitives that so far were only subject to theoretical cryptanalytic examination;
2. in a area such as cryptography where there is a large need to be able to trust an object (here a transformed cryptographic primitive), one more way to test the object is likely welcomed so as to (in case the object passes the test) increase our trust in the object.

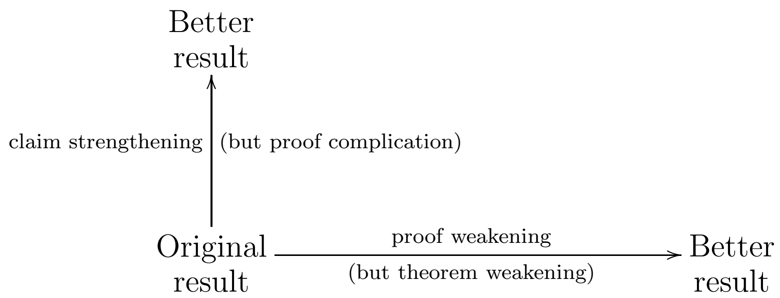
10.8 Proposal: claim strengthening and proof weakening

10.11 Proposal.

Idea We propose to improve transformations of cryptographic primitives by means of strengthening the theorems' claims or, inversely, weakening the theorems' proofs.

Claim strengthening Given a transformation theorem claiming that a cryptographic primitive P_1 can be transformed into a cryptographic primitive P_2, we may be able to strengthen the claim by changing the transformation so that P_2 has extra properties (such as length regularity or even length preservation).

Proof weakening Given a transformation proof showing that a cryptographic primitive P_1 can be transformed into a cryptographic primitive P_2 with extra properties, we may be able to weaken the proof by changing the transformation so that P_2 does not need to have the extra properties.



Comments

Claim strengthening Strengthening a theorem's claim is a conventional goal in mathematics and computer science because one usually searches for the "best" results, often in the sense of stronger. The price to pay is that the theorem's proof has to be made more complicated along with the theorem's claim.

For example in sections 7.7, 7.8 and 7.9 we proved that indistinguishability from random implies three other security notions, where our initial idea was to do the proof only for stream ciphers C_G induced by pseudorandom generators G, but then we realised that the implications are more meaningful (indistinguishability from random is somewhat new and so needs to be justified) if strengthened to "arbitrary" (actually, length regular) stream ciphers. The price that we paid was:

1. to add the assumption of length regularity in the implications;
2. to discuss the meaning of length regularity as a modest security notion;
3. to prove that the implications are false without length regularity.

Proof weakening Weakening a theorem's proof is an unconventional goal in mathematics but more accepted in teaching (in a loose sense including talks, textbooks and popularisation works) because one may have to lower the difficulty level of a proof to better pass a message. The price to pay is that the theorem's claim has to be weakened along with the proof.

For example we simplified proof 4.16 of theorem 4.15 by weakening formula (4.10)

1. from $\Pr[\ldots] = \varepsilon/2$ to $\Pr[\ldots] \leq \varepsilon/2$ (by replacing $=$ by $\leq$), which allowed to skip the analysis of cases (4.11) and (4.14);
2. from $\Pr[\ldots] \leq \varepsilon/2$ to $\Pr[\ldots] \leq 1/2$ (by taking $\varepsilon := 1$), which allowed to simplify the notation and the calculations.

The price that we paid was that we had to weaken the theorem and its security notions according to the proof.

Benefit We see the following benefits of this proposal.

Claim strengthening Strengthening a theorem's claim may produce a better result for research proposes.

Proof weakening Weakening a theorem's proof may produce a better result for teaching.

10.9 Proposal: transformation of cryptographic protocols

10.12 Proposal.

Idea We propose to study transformations of cryptographic protocols P as done with cryptographic primitives, namely treating P as a "black-box" for which:

1. we cannot "look inside" P;
2. we can only access P through its two "interfaces"
   (a) $x \stackrel{\text{in}}{\to} P$ for input, as in "we give x as an input to P";
   (b) $P \stackrel{\text{out}}{\to} y$ for output, as in "we receive y as an output from P".

Comments For example, let us consider the following simple cryptographic protocol P, where E is an encryption algorithm, k_A is Alice's key and k_B is Bob's key.

(1) The cryptographic protocol receives x as input.
(2) Alice computes and sends $E(k_A, x)$ to Bob.
(3) Bob computes and sends $E(k_B, E(k_A, x))$ to Alice.
(4) The cryptographic protocol gives $E(k_B, E(k_A, x))$ as output.

Then:

1. saying that we cannot "look inside" P means that we cannot change the inner-workings of P, for example we cannot
   (a) change step (1) to "The cryptographic protocol receives x and y as inputs";
   (b) drop step (3);
   (c) add an new step between steps (2) and (3);
2. saying that we can only access P through its "interfaces" $x \overset{\text{in}}{\to} P$ and $P \overset{\text{out}}{\to} y$ means that the only two things that we can do with P is to
   (a) give a value x as input to the P, that is $x \overset{\text{in}}{\to} P$;
   (b) receive a value y as output from P, that is $P \overset{\text{out}}{\to} y$.

The way in which we can use P is limited but still we can do some transformations of P, for example:

1. we can compose P with some cryptographic protocol, for example with P itself, getting the new cryptographic protocol $P \circ P$ that on input x outputs
$$E\big(k_B, E\big(k_A, E(k_B, E(k_A, x))\big)\big),$$
or schematically,
$$x \overset{\text{in}}{\to} P \overset{\text{out}}{\to} E(k_B, E(k_A, x)) \overset{\text{in}}{\to} P \overset{\text{out}}{\to} E\big(k_B, E\big(k_A, E(k_B, E(k_A, x))\big)\big).$$
2. we can
   (a) pre-process with a function f the input x of P before giving it to P, which means that instead of giving x we actually give $f(x)$;
   (b) post-process with a function g the output y of P, which means that instead of receiving y we actually receive $g(y)$;

   getting the new cryptographic protocol $g \circ P \circ f$ that on input x outputs
$$g\big(E\big(k_B, E(k_A, f(x))\big)\big),$$
   or schematically,
$$x \overset{\text{pre}}{\to} f(x) \overset{\text{in}}{\to} P \overset{\text{out}}{\to} E\big(k_B, E(k_A, f(x))\big) \overset{\text{post}}{\to} g\big(E\big(k_B, E(k_A, f(x))\big)\big).$$

Benefit We see the following benefit of this proposal: to apply to transformation of cryptographic protocols the ideas that we developed for, and the lessons that we learned from, transformation of cryptographic primitives.

10.10 Proposal: recasting cryptographic concepts as topological concepts

10.13 Proposal. Often security notions have definitions similar to the probability that a reasonable algorithm "sees" the difference between an "ideal" object $X(1^n)$ (for example a truly random binary string of length n) and a "real" object $Y(1^n)$ (for example the prefix of length n of the stream of a pseudorandom generator) is negligible, that is

$$\forall A, p\ \exists N\ \forall n > N\ |\Pr A(X(1^n)) - \Pr A(Y(1^n))| < 1/p(n), \tag{10.1}$$

where

1. A ranges over the polynomial-time probabilistic algorithms;
2. p ranges over the positive polynomials;
3. N and n range over $\mathbb{N}$.

Stefan Kahrs suggested the idea of:

1. giving topologies to sets used in cryptography, for example to give to the set $\mathbb{R}^{\mathbb{N}}$ (of the functions $f\colon \mathbb{N} \to \mathbb{R}$) the topology having as a base the set of "balls" of the form

   $$B_p(f) := \{g \in \mathbb{R}^{\mathbb{N}} \mid \exists N\ \forall n > N\ |f(n) - g(n)| < 1/p(n)\},$$

   where p is a positive polynomial;

2. recasting (10.1) as saying that for all reasonable algorithms A, the "points" $X(1^n)$ and $Y(1^n)$ are topological indistinguishable, or more precisely that for all polynomial-time probabilistic algorithms A, we have that $\Pr A(X(1^n))$ and $\Pr B(Y(1^n))$ are topologically indistinguishable;

3. asking questions about the continuity of transformations of cryptographic primitives;

4. determining how low are these topological spaces in the hierarchy of separation axioms T_0–T_6;

5. using notions from topology, for example compactness;

6. using results from topology, for example that the composition of continuous functions is continuous;

7. abstract from quantifiers (such as $\exists N\ \forall n > N$) by moving them from the cryptographic foreground to the topological background (as in the definition of $B_p(f)$).

Eerke Boiten suggested that a topological approach to the transformation of cryptographic primitives may link "ideal" secure cryptographic primitives (for example one-way functions) and "real" maybe-secure cryptographic primitives (for example the Secure Hash Algorithm 3 SHA3), relying on the premise that "real" maybe-secure cryptographic primitives approximate "ideal" secure cryptographic primitives: if a transformation T, of a secure theoretic cryptographic primitive P_1 into another secure theoretic cryptographic primitive P_2, is continuous, then T should also transform a "real" maybe-secure applied cryptographic primitive P_1' into another "real" maybe-secure applied cryptographic primitive P_2'; in symbols, if

$$(*)\ \underbrace{T(P_1) = P_2}_{T \text{ transforms } P_1 \text{ into } P_2}, \qquad (\dagger)\ \underbrace{\forall P, P'\ (P \approx P' \ \Rightarrow\ T(P) \approx T(P'))}_{T \text{ is continuous}}, \qquad (\ddagger)\ \underbrace{P_i' \approx P_i}_{P_i' \text{ approximates } P_i},$$

then

$$T(P_1') \overset{(\dagger)(\ddagger)}{\approx} T(P_1) \overset{(*)}{=} P_2 \overset{(\ddagger)}{\approx} P_2',$$

so (assuming that $\approx$ is transitive)

$$\underbrace{T(P_1') \approx P_2'}_{T \text{ approximately transforms } P_1' \text{ into } P_2'}.$$

10.11 Conclusion

10.14. In this chapter we presented some proposals to study more transformations of cryptographic primitives following these research lines:

1. completion of the graph;
2. addition of new vertices to the graph;
3. basing the construction of cryptographic primitives on indistinguishable-from-random stream ciphers;
4. extraction of numeric/computational content;
5. composition and decomposition of transformations;
6. empirically test implemented candidates to cryptographic primitives obtained by transformations;
7. improving transformation claims or transformation proofs;
8. transformation of cryptographic protocols;
9. recast cryptographic concepts as topological concepts.

Chapter 11

Proof presentation of transformation of cryptographic primitives

11.1 Introduction

11.1. Similarly to what we said in paragraph 10.1, in section 1.2 we identified a problem in cryptography: definitions, theorems and proofs in cryptography are often so complicated that some incorrect "proofs" go unnoticed. In sections 1.3, 1.4, 1.5 and 1.6 we proposed a solution for the problem. The solution is divided into components and one of the components is proof presentation. In section 1.5 we argued that proof presentation helps solve the problem by helping to check the correctness of proofs, helping to understand proofs and increasing the audience for proofs. So, ultimately, in the "big picture" of things, our work devoted to proof presentation hopefully contributes to the solution of the problem.

11.2. To be more concrete, we give three examples of small insights that we developed.

1. Security proofs of transformations of cryptographic primitives can easily become complex because of

   (a) intricate definitions of cryptographic primitives and security notions;

   (b) several theories used in the security proofs;

   (c) delicate syntactical rewriting of formulas;

   (d) large number of objects mentioned in the proof such as cryptographic primitives, security notions, security parameters, algorithms, formulas, calculations and divisions in cases;

   and so on, thus being organised and having a clear structure helps managing the complexity of proofs. Let us give as examples some simple, yet effective, small tricks to help organising and structuring proofs:

(a) stating at the beginning of the proof its premises/assumptions and its conclusions/goals (because we may lose sight of what we know and what we want to know in a long proof, find it convenient to be reminded, or simply to introduce the notation that will be used such as the one-way binary-string function being named f and the algorithm that tries to break its security being named A);

(b) using text structures (such as lists, indentation, tables and text division into paragraphs) to organise content;

(c) dividing the proof into lemmas, claims, assumptions and goals (to "divide and conquer" and also because lemmas and claims may be provable in isolation);

(d) giving labels to assumptions, goals, claims and (parts of) formulas, and then refer to them by their labels instead of repeating them (such as "by formula (9)" instead of "by $\forall A, B\ \Pr[A(\ldots) = B(\ldots)] \in \mathcal{N}$") but avoiding so many labels that it becomes difficult to keep track of them (it may be difficult to find the objects with labels $(1), (2), (3), \ldots, (*), (\dagger), (\S), \ldots, (\alpha), (\beta), (\gamma), \ldots$ and $(P), (P'), (P''), \ldots$ in the middle of the text);

(e) using easy-to-remember notation (for example algorithms attacking cryptographic primitives P_1, P_2 and P_3 are better denoted by A_1, A_2 and A_3 than by X, X' and X'', or even worse, by $\aleph$, β and c);

(f) being careful with the writing style (for example mixing the wordings "from P we get Q" and "we get Q from P" can be confusing because they express implications $P \Rightarrow Q$ and $Q \Leftarrow P$ where the arrows have opposing directions, or even worse, writing the nonsensical "due to P, since Q, we have R, because of S, by T");

(g) relying on conventions and abbreviations to shorten a formula (such as $\Pr[X = Y]$ meaning $\Pr[x = y : x \leftarrow X, y \leftarrow Y]$) but not to the point of becoming difficult to interpret the formula (it may not be clear whether $\Pr[X = X]$ means $\Pr[x = x : x \leftarrow X]$ or $\Pr[x = x' : x \leftarrow X, x' \leftarrow X]$).

The listed tricks are not specific to cryptography but become particularly important there to manage complexity. A list of this nature would never be exhaustive but the examples given should be enough to explain the kind of tricks that we have in mind.

2. There is a tension between producing a deep proof and producing a clear proof because they are often orthogonal aims. Let us give as examples some simple, yet effective, small tricks to help writing clear proofs:

   (a) the type of proof is often decided by the audience of the proof, namely

      i. for research-like activities, such as publishing a research article or giving a research talk in a specialised conference, a deep proof is better;

      ii. for teaching-like activities, such as writing a textbook for students or giving a popularisation talk for laypersons, a clear proof is better;

(b) if space and time allows it, it is possible not to have to make a choice between the two proofs by presenting both of them as complements of one another;

(c) sometimes a deep proof can also be made clear by rewriting the proof using high-level/abstract concepts such as $\lim_{n\to\infty} x_n = l$ instead of low-level/concrete objects such as

$$\forall\varepsilon > 0\ \exists N \in \mathbb{N}\ \forall n \in \mathbb{N}\ (n > N \ \Rightarrow\ |x_n - l| < \varepsilon);$$

(d) sometimes a deep proof only lacks clarity because it is badly written, which could be substantially improved if we were allowed the time to think more about the proof so as to get a better understanding of it and therefore an enhanced ability to present the proof well (unfortunately, we find ourselves too often in a situation where time urges because we need to submit that article by the deadline of the next conference, we need that publication to improve our odds of getting a grant or a job promotion, or we are too busy teaching, writing referee reports or taking care of administrative tasks).

3. Sometimes the graphical appearance of a written proof makes a difference in understanding the proof. Let us give as examples some simple, yet effective, small tricks to help writing more-graphical proofs:

(a) drawing a diagram that captures the statement that we want to prove (for example, De Morgan's law $\overline{A \cap B} = \overline{A} \cup \overline{B}$ becomes evident when we draw it as a Venn diagram);

(b) instead of being bounded by the usual linear presentation of written text, we may want to take advantage of the two-dimensional shape of paper and write, say, in a tree-like shape (for example when proving the zero-product property $xy = 0 \ \Rightarrow\ x = 0 \lor y = 0$ by considering the three cases $x < 0$, $x = 0$ and $x > 0$, which in turn are divided into the three subcases $y < 0$, $y = 0$ and $y > 0$);

(c) if some parts of a proof are under the scope of other parts, this can be made apparent by using indentation to indicate scope as usually done with programming languages (for example

```
Assume P.            for (i = 1; i <= n; i++) {
  Then P.              sum += i;
So P ⇒ P.            }
```

where the second lines are under the scope of the first lines).

(d) it often helps to present a statement by a figure (such as a commutative diagram expressing $f \circ g = g \circ f$) as a complement to its presentation in words and symbols (such as "functions f and g commute under composition in the sense of $f \circ g = g \circ f$"), or at least present a figure-like formula (such as $x \overset{g}{\mapsto} \cdots \overset{f}{\mapsto} y \overset{g}{\hookleftarrow} \cdots \overset{f}{\hookleftarrow} x$).

11.3. We already saw in previous chapters the 11 proof presentations listed in the following table.

Section	Proof presentation
1.5	Proof idea
1.5	Visualisation of a proof
1.5	Emphasising proof structure
5.2	Bad cryptographic definition
5.3	Incorrect "proof"
5.4	Notation improvement
6.4	Proof idea
7.4	Schematic proof
7.5	Wedding-cake notation
8.4	Carving out a theory
9.4	Indirect proof

In this chapter we:

1. recover some of these proof presentations;

2. present some new proof presentations.

Each proof presentation is turned into a case study with the following 6-part structure.

Introduction: idea Idea of the problem and solution.

Problem: number theory Easy number-theoretic example of the problem.

Solution: number theory Easy number-theoretic example of the solution.

Problem: cryptography Difficult cryptographic example of the problem.

Solution: cryptography Difficult cryptographic example of the solution.

Conclusion: lesson Lesson learned about the problem and solution.

11.4. In this chapter all work is ours except:

1. the triangular inequality and its proof without using intervals in section 11.2 (which are likely "folklore");

2. the notion of eventual domination for functions in section 11.5 (which is likely "folklore").

11.2 Proof presentation: proof idea

11.5 Case study.

Introduction: idea Sometimes there is a clear idea behind a proof that is not clear from the written proof, and it becomes a non-trivial exercise for the reader to recover the idea from the written proof. We think that this happens in three steps:

1. the prover uses his/her intuition to produce a clear idea for the proof;
2. the prover writes down the proof in a semi-formal or formal language while taking care of technical details, and in the process the idea becomes "buried" under the language and the technicalities;
3. the reader has to reverse the prover's second step in order to expose the prover's first step and so recover the idea.

We are not claiming that the prover's second step is not worthwhile, to the contrary, it is necessary to produce a proof formalised enough to be checked for correctness. What we are claiming is that something important is lost in the second step, namely the idea that would illuminate the reader in three ways:

1. by explaining why the proof is the way it is (instead of being the "pull of a rabbit out of a hat" criticised by Dan Grundy (Grundy 2008, page 20));
2. by explaining why the result is true (instead of just establishing that the result is true);
3. by working as a mnemonic for the proof (we have good memory for clever ideas but not for long and arbitrary-looking manipulations of symbols).

Problem: number theory Let us consider the following proposition (the triangular inequality) and its proof.

Proposition. We have $\forall x, y \in \mathbb{R}\ |x+y| \leq |x| + |y|$. □

Proof. Let $x, y \in \mathbb{R}$ be arbitrary. We have:

$$\begin{aligned} x \leq |x| \wedge x \geq -|x| \wedge y \leq |y| \wedge y \geq -|y| \ &\Rightarrow \\ x + y \leq |x| + |y| \wedge x + y \geq -(|x| + |y|) \ &\Rightarrow \\ |x + y| \leq |x| + |y|. & \end{aligned}$$ □

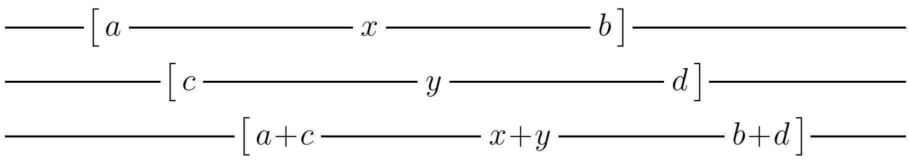
There is a simple idea for the proof but the idea is not easily seen from the proof: knowing that x is in the interval $[-|x|, |x|]$ and that y is in the interval $[-|y|, |y|]$, we can calculate an interval $[-(|x| + |y|), |x| + |y|]$ where $x + y$ is, concluding $|x + y| \leq |x| + |y|$. Next we are going to develop further this idea.

Solution: number theory Let $x \in [a, b]$ and $y \in [c, d]$, and let us define an operation $+$ on intervals $[a, b]$ and $[c, d]$ such that $(*)$ $x+y \in [a, b]+[c, d]$. There are two natural definitions, which give the same result and for which $(*)$ holds:

1. $[a, b] + [c, d] := [a + c,\ b + d]$;
2. $[a, b] + [c, d] := \{r + s \in \mathbb{R} \mid r \in [a, b],\ s \in [c, d]\}$;

(assuming $a \leq b$ and $c \leq d$ to avoid pathological cases such as $[0, -1]+[0, 1] := [0 + 0,\ -1 + 1] = [0, 0] \neq \emptyset$ by point 1 but $[0, -1] + [0, 1] := \{r + s \in \mathbb{R} \mid$

$r \in \emptyset,\ s \in [0,1]\} = \emptyset$ by point 2, where $[0,-1] := \{r \in \mathbb{R} \mid 0 \le r \le -1\} = \emptyset$).
This is illustrated in the figure below.



In the next proof we make use of this natural addition of intervals to get a proof displaying a clear idea.

Proof.

1. We define
$$[a,b] + [c,d] := [a+c,\ b+d],$$
so, for all $x, y \in \mathbb{R}$,
$$x \in [a,b] \ \wedge\ y \in [c,d] \ \Rightarrow\ x + y \in [a+c,\ b+d].$$

2. For arbitrary $x, y \in \mathbb{R}$, we have
$$\begin{aligned} x \in [-|x|, |x|] \ \wedge\ y \in [-|y|, |y|] \ &\Rightarrow \\ x + y \in [-|x|, |x|] + [-|y|, |y|] = [-(|x| + |y|),\ |x| + |y|] \ &\Rightarrow \\ |x + y| \le |x| + |y|. & \end{aligned}$$ ☐

Problem: cryptography Let us consider proof 7.34 (summarised below).

Proposition. For all length-regular stream ciphers C, if C is indistinguishable from random, then C is semantically secure. ☐

Proof. The stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$ being indistinguishable from random means
$$\begin{gathered} \forall A, A' \ \Pr A'\big(E(K(1^n), A(1^n)_1), 1^n, A(1^n)_2\big) - \\ \Pr A'(U_{|E(K(1^n), A(1^n)_1)|}, 1^n, A(1^n)_2) \in \mathcal{N}. \end{gathered}$$

The stream cipher C being semantically secure means
$$\begin{gathered} \forall B' \ \exists C' \ \forall B, f, g \\ \Pr\big[B'\big(E(K(1^n), B(1^n)_1), f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] - \\ \Pr\big[C'\big(1^{|B(1^n)_1|}, f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] \in \mathcal{N}. \end{gathered}$$

Taking $A(x) := (B(x)_1, B(x))$ and $A'(x, y, z) := \mathrm{Tr}[B'(x, f(z, y), y, z_2) = g(z, y)]$ in the premise, we get
$$\begin{gathered} \forall B', B, f, g \\ \Pr\big[B'\big(E(K(1^n), B(1^n)_1), f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] - \\ \Pr\big[B'\big(U_{|E(K(1^n), B(1^n)_1)|}, f(B(1^n), 1^n), 1^n, B(1^n)_2\big) = g(B(1^n), 1^n)\big] \in \mathcal{N}. \end{gathered}$$

We have $|E(0, 1^{|B(1^n)_1|})| = |E(K(1^n), B(1^n)_1)|$ by the length regularity of C, so taking $C'(w, x, y, z) := B'(U_{|E(0,w)|}, x, y, z)$, we get the conclusion. ☐

This proof has two main ideas:

1. moving from an algorithm A' computing a truth value 1 to an algorithm B' computing a function g by $A'(x,y,z) := \mathrm{Tr}[B'(x, f(z,y), y, z_2) = g(z,y)]$;
2. hardwiring of a uniform random variable $U_{|E(0,w)|}$ in the algorithm C' by $C'(w,x,y,z) := B'(U_{|E(0,w)|}, x, y, z)$.

Moreover, the proof as it is written is:

1. good to check for correctness (not so much the summarised proof above but the more detailed proof 7.34);
2. bad to show the two ideas.

We see this as problem that we should fix.

Solution: cryptography The best way to fix the problem identified above is to give a new proof that is both:

1. good to check for correctness;
2. good to show the two ideas.

Because we lack such an ideal proof, we give a complementary proof that is:

1. bad to check for correctness;
2. good to show the two ideas.

Proof. The proof has the following two main ideas.

1. Indistinguishability from random essentially has the form

$$\forall A' \;\; \Pr[A'(\ldots) = 1] - \Pr[A'(\ldots) = 1] \in \mathcal{N},$$

so it talks about an algorithm A' computing the truth value 1. Semantic security essentially has the form

$$\forall B' \; \exists C' \; \forall g \;\; \Pr[B'(\ldots) = g(\ldots)] - \Pr[C'(\ldots) = g(\ldots)] \in \mathcal{N},$$

so it talks about algorithms B' and C' computing a function g. To move from indistinguishability from random to semantic security, we have to solve the following problem: how to "upgrade" from an algorithm A' computing the truth value 1 to an algorithm B' (and C') computing a function g, in the sense of, given B' (and g), to choose an A' such that

$$A'(\ldots) = 1 \;\Leftrightarrow\; B'(\ldots) = g(\ldots).$$

When the problem is phrased this clearly, the solution is fairly clear:

$$A(\ldots) := \mathrm{Tr}[B'(\ldots) = g(\ldots)].$$

2. The previous idea leads us to an intermediate formula essentially of the form

$$\forall B' \; \forall g \;\; \Pr[B'(E(\ldots),\ldots) = g(\ldots)] - \Pr[B'(U_{\ldots},\ldots) = g(\ldots)] \in \mathcal{N},$$

   where now we made explicit the inputs $E(\ldots)$ and $U_{\ldots}$ because they will play a role. Semantic security essentially has the form

$$\forall B' \; \exists C' \; \forall g \;\; \Pr[B'(E(\ldots),\ldots) = g(\ldots)] - \Pr[C'(\ldots) = g(\ldots)] \in \mathcal{N},$$

   so it asks to construct an algorithm C'. To move from the intermediate formula to semantic security, we have to solve the following problem: how to construct the algorithm C', in the sense of, given an algorithm B', to produce an algorithm C' such that

$$C'(\ldots) = B'(U_{\ldots},\ldots).$$

   When the problem is phrased this clearly, the solution is clear:

$$C'(\ldots) := B'(U_{\ldots},\ldots),$$

   where $U_{\ldots}$ is "hardwired" in B' to create C', which can be done because C' is intended to be a probabilistic algorithm and so it can create its own $U_{\ldots}$ using its own "internal coin tosses".

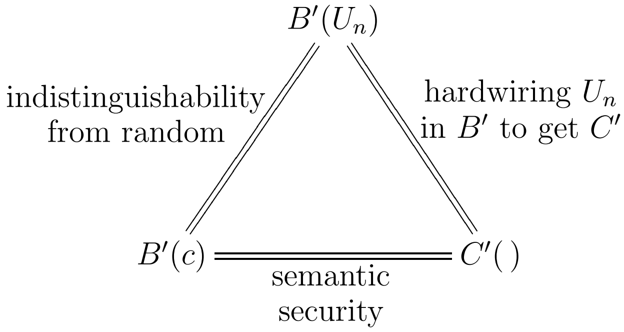
Let us summarise, less precisely but more clearly, these two ideas.

1. Indistinguishability from random talks about A' computing 1 while semantic security talks about B' computing g, and we can move from the former to the latter by $A' := \mathrm{Tr}[B' = g]$.
2. After applying the previous idea:
   (a) indistinguishability from random now talks about B' "not seeing" the difference between a ciphertext c and a random string U_n, that is $B'(c) = B'(U_n)$;
   (b) semantic security talks about what can be computed by B' seeing a ciphertext c can also be computed by some C' without seeing the ciphertext c, that is $B(c) = C'(\,)$.

   We can move from the former to the latter by taking $C'(\,) := B'(U_n)$:

$$\begin{array}{rll} \text{indistinguishability from random:} & B'(c) = B'(U_n), & \\ \text{hardwiring } U_n \text{ in } B \text{ to get } C': & \phantom{B'(c) = {}} B'(U_n) = C'(\,), & \\ \text{semantic security:} & B'(c) = \phantom{B'(U_n) = {}} C'(\,). & \end{array}$$

or, more diagrammatically,



where the double lines denote equality. □

To be sure, we are not arguing that this new proof should replace the original proof (because the new proof is not formalised enough to be checked for correctness), only that the new proof complements the original proof by showing more clearly the two ideas behind the original proof.

Conclusion: lesson Sometimes a proof follows a clear idea not easily read from the proof. This deprives the reader from an explanation to the proof, to the truth of the result, and a mnemonic for the proof. Instead of forcing the reader to non-trivially extract the idea from the proof, we propose that the proof should be rewritten in a way that exposes the idea or at least the idea should be given as a complement to the proof.

11.3 Proof presentation: schematic proof and proof reorganisation

11.6 Case study.

Introduction: idea Sometimes proofs have a structure more complicated than needed. A simple example is a direct proof of $P \Rightarrow Q$ versus a direct proof unnecessarily combined with a proof by contradiction.

Direct	Direct and contradiction
Assume P. From P get… so Q.	Assume P. Aiming at a contradiction, assume $\neg Q$. From P get… so Q. From Q and $\neg Q$ we get a contradiction $\perp$. So Q.
$\begin{array}{c} P \\ \vdots \\ Q \end{array}$	$\dfrac{\dfrac{\begin{array}{c} P \\ \vdots \\ Q \end{array} \quad \neg Q}{\perp}}{Q}$

Proof organisation is the act of changing the arguments and their order in proofs to make proofs clearer and/or shorter. It can be achieved in two ways (Dijkstra 2000, page 22):

1. the "bad" way, which consists in omitting details that help understand the proof;
2. the "good" way, which consists in eliminating redundancies in the proof and finding shorter "proof paths".

In order to do proof reorganisation in the "good" way, it is often convenient to express the proof schematically to better reveal its structure. There are various schematic notations, for example:

1. Wim H. J. Feijen's notation (Grundy 2008, page 17) (which works well for linear proofs) with the format like

$$\begin{array}{ll} & \neg(P \wedge Q) \\ \Leftrightarrow & \quad \{\,\text{De Morgan's law}\,\} \\ & \neg P \vee \neg Q \\ \Leftrightarrow & \quad \{\,\text{Commutativity}\,\} \\ & \neg Q \vee \neg P \end{array}$$

2. the natural-deduction notation (Gentzen 1935, page 188) (which works well for unidirectional trees) with the format like

$$\cfrac{\cfrac{\neg P \qquad \neg Q}{\neg P \wedge \neg Q}\ \wedge\ \text{introduction}}{\neg(P \vee Q)}\ \text{De Morgan's law}$$

3. Daniel J. Velleman's notation (Velleman 2006, page xii) (which is typographically convenient and programming-like but not so graphical) with the format like

$$\begin{array}{l} \text{Assume } P. \\ \quad \text{Thus } P. \\ \quad \text{Thus } P \vee Q. \\ \text{Thus } P \Rightarrow P \vee Q. \end{array}$$

4. "our" notation (which works well for unidirectional and bidirectional trees) with the format like

$$\begin{tikzcd} \neg P \arrow[dr, Rightarrow, "\text{Weakening}"'] & & \neg Q \arrow[dl, Rightarrow, "\text{Weakening}"] \\ & \neg P \vee \neg Q \arrow[d, Leftrightarrow, "\text{De Morgan's law}"] & \\ & \neg(P \wedge Q) & \end{tikzcd}$$

Problem: number theory Let us consider the following proposition and its proof.

Proposition. We have $\forall x \in \mathbb{R}\ \forall n \in \mathbb{N} \setminus \{0, 1\}\ (x^n = x \ \Rightarrow\ x \in \{-1, 0, 1\})$. $\square$

Proof. Let $\mathbb{O} := \{1, 3, 5, \ldots\}$ denote the set of positive odd numbers and $\mathbb{E} := \{2, 4, 6, \ldots\}$ denote the set of positive even numbers. We have that $x^n = x$ is equivalent to $x(x^{n-1} - 1) = 0$ (notice that x^{n-1} is defined for $x = 0$ because $n - 1 \geq 1$), which is equivalent to $x = 0$ or $x^{n-1} - 1 = 0$. If $n \in \mathbb{E}$, then $n - 1 \in \mathbb{O}$, so $x^{n-1} - 1 = 0$ is equivalent to $x^{n-1} = 1$ that is equivalent to $x = 1$. If $n \in \mathbb{O}$, then $n - 1 \in \mathbb{E}$ because $n \neq 1$, so $x^{n-1} - 1 = 0$ is equivalent to $x^{n-1} = 1$ that is equivalent to $x = \pm 1$. This proof is essentially presented diagrammatically below.

$$\underline{x^n = x}$$

$$\Big\Updownarrow \ n - 1 \geq 1$$

$$\underline{x(x^{n-1} - 1) = 0}$$

$$\Big\Updownarrow \ \text{zero-product property}$$

$$x = 0 \vee \underline{x^{n-1} - 1 = 0}$$

$$\Big\Updownarrow \ \text{logic}$$

$$\underbrace{(\underline{n \in \mathbb{E}} \wedge x^{n-1} - 1 = 0)} \vee \underbrace{(\underline{n \in \mathbb{O}} \wedge x^{n-1} - 1 = 0)}$$

$$\Big\Updownarrow \qquad\qquad\qquad\qquad \Big\Updownarrow \ n \neq 1$$

$$\underline{n - 1 \in \mathbb{O}} \wedge \underline{x^{n-1}} - 1 = 0 \qquad\qquad \underline{n - 1 \in \mathbb{E}} \wedge \underline{x^{n-1}} - 1 = 0$$

$$\text{arithmetic} \Big\Updownarrow \qquad\qquad\qquad\qquad \Big\Updownarrow \text{arithmetic}$$

$$n - 1 \in \mathbb{O} \wedge \underline{x^{n-1}} = 1 \qquad\qquad n - 1 \in \mathbb{E} \wedge \underline{x^{n-1}} = 1$$

$$\text{arithmetic} \Big\Downarrow \qquad\qquad\qquad\qquad \Big\Downarrow \text{arithmetic}$$

$$x = 1 \qquad\qquad\qquad\qquad x = \pm 1$$

$\square$

From the prose proof it is not easily noticeable, but from the diagrammatic proof we see more easily that the proof is wasteful in three ways:

1. from the first underlined part (which is $x^n = x$) to the second underlined part (which is $x(x^{n-1} - 1) = 0$) there is a "large" step (which is $x^n = x \ \Leftrightarrow\ x^n - x = 0 \ \Leftrightarrow\ xx^{n-1} - x = 0 \ \Leftrightarrow\ x(x^{n-1} - 1) = 0$), but from the first underdotted parts (which are $x^{n-1} - 1 = 0$) to the second underdotted parts (which are $x^{n-1} = 1$) there is only a "small" step, so we can avoid the "small" step by rewriting the third underlined part (which is $x^{n-1} - 1 = 0$) as $x^{n-1} = 1$;

2. the underdotted parts (which are $x^{n-1} - 1 = 0$ and $x^{n-1} = 1$) are the same on both branches of the proof, so we can avoid this repetition by moving the underdotted parts to before the branching;
3. the first underwaved parts (which are $n \in \mathbb{E}$ and $n \in \mathbb{O}$) are introduced but what is used are the second underwaved parts (which are $n - 1 \in \mathbb{O}$ and $n - 1 \in \mathbb{E}$), so we can skip the first underwaved parts.

Solution: number theory Now we repeat the diagrammatic proof with the three improvements mentioned above implemented.

Proof.

$$\begin{array}{c}
x^n = x \\
\Big\Updownarrow \; n - 1 \geq 1 \\
x(x^{n-1} - 1) = 0 \\
\Big\Updownarrow \; \text{zero-product property} \\
x = 0 \lor x^{n-1} = 1 \\
\Big\Updownarrow \; \text{logic, } n \neq 1 \\
\underbrace{(n - 1 \in \mathbb{O} \land x^{n-1} = 1)} \lor \underbrace{(n - 1 \in \mathbb{E} \land x^{n-1} = 1)}
\end{array}$$

$$\begin{array}{ccc}
\text{arithmetic} \swarrow & & \searrow \text{arithmetic} \\
x = 1 & & x = \pm 1
\end{array}$$

□

We see that the number of edges of the diagram is reduced from 10 to 6, which is a considerable shortening.

Problem: cryptography Let us consider proof 6.8. The way in which we wrote it is already organised, so let us schematically present a slightly less organised variant below for example purposes.

Proposition. Let f be a fixed collision-resistant binary-string function. For all length-nondecreasing binary-string functions g, if g is one-way, then $f \circ g$ is one-way. □

Proof.

$$\begin{array}{c}
\underbrace{\begin{array}{c} \forall A \;\; \Pr[\underline{|A(1^n)| \geq n} \land \\ A(1^n)_1 \neq A(1^n)_2 \land \\ f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N} \end{array} \qquad \begin{array}{c} \forall B \; \Pr\big[g\big(B(g(U_n), 1^n)\big) = \\ g(U_n)\big] \in \mathcal{N} \end{array}} \\
\Big\Downarrow \; \begin{array}{c} \Pr[Q \lor R] \leq \Pr Q + \Pr R \\ f, g \in \mathcal{N} \;\Rightarrow\; f + g \in \mathcal{N} \\ f \leq g \in \mathcal{N} \;\Rightarrow\; f \in \mathcal{N} \end{array} \\
\forall A, B \;\Pr\big[\big(\underline{|A(1^n)| \geq n} \land A(1^n)_1 \neq A(1^n)_2 \land \\
f(A(1^n)_1) = f(A(1^n)_2)\big) \lor g\big(B(g(U_n), 1^n)\big) = g(U_n)\big] \in \mathcal{N}
\end{array}$$

$$\Big\Downarrow\ \begin{array}{c} A(x) := \big(g\big(C(f \circ g(U_{|x|}), x)\big), g(U_{|x|})\big) \\ B(x,y) := C(f(x), y) \end{array}$$

$$\begin{array}{c}
\forall C\ \Pr\big[\big(\underline{\big|g\big(C(f \circ g(U_n), 1^n)\big)\big| + |g(U_n)| \geq n} \ \wedge \\
g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \\
f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big) \ \vee \\
g\big(C(f \circ g(U_n), 1^n)\big) = g(U_n)\big] \in \mathcal{N}
\end{array}$$

$$\Big\Downarrow\ \underline{|g(U_n)| \geq n}$$

$$\begin{array}{c}
\forall C\ \Pr\big[\big(g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \\
f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big) \ \vee \\
g\big(C(f \circ g(U_n), 1^n)\big) = g(U_n)\big] \in \mathcal{N}
\end{array}$$

$$\Big\Downarrow\ \begin{array}{c} (x \neq x' \ \wedge \ f(x) = f(x')) \ \vee \\ x = x' \ \Leftrightarrow \ f(x) = f(x') \end{array}$$

$$\forall C\ \Pr\big[f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big] \in \mathcal{N} \qquad \square$$

We can see that the proof is complicated in two ways:

1. the first three underlined parts (which are $|A(1^n)| \geq n$, $|A(1^n)| \geq n$ again and $\cdots + |g(U_n)| \geq n$) are "dragged" along the proof only to be eliminated by the fourth underlined part (which is $g(U_n) \geq n$) but this elimination could be done sooner for simplicity;

2. we move from the first underdotted part (which is $\forall A, B \ldots$) to the third underdotted part (which is $\forall C \ldots$) by the instantiation in the second underdotted part (which is $A(x) := \cdots$ and $B(x,y) := \cdots$) but this instantiation is actually two instantiations (one from A and another one for B) that act disjointly in the first underlined part (which is a disjunction in which the left disjunct mentions A but not B and the right disjunct mentions B but not A), so they could be done separately for simplicity.

<ins>Solution: cryptography</ins> By simplifying the two complications mentioned, we obtain proof 6.8, which we summarise below diagrammatically for ease of comparison with the above variant.

Proof.

$$
\begin{array}{c}
\forall A \ \Pr[|A(1^n)| \geq n \ \wedge \\
A(1^n)_1 \neq A(1^n)_2 \ \wedge \\
f(A(1^n)_1) = f(A(1^n)_2)] \in \mathcal{N}
\end{array}
$$

$$
\begin{array}{c} A(x) := \\ \big(g\big(C(f \circ g(U_{|x|}), x)\big), \\ g(U_{|x|})\big) \end{array} \Bigg\Downarrow
$$

$$
\begin{array}{cc}
\begin{array}{c}
\forall C \ \Pr\big[\big|g\big(C(f \circ g(U_n), 1^n)\big)\big| + |g(U_n)| \geq n \ \wedge \\
g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \\
f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big] \in \mathcal{N}
\end{array}
&
\begin{array}{c}
\forall B \ \Pr\big[g\big(B(g(U_n), 1^n)\big) = \\
g(U_n)\big] \in \mathcal{N}
\end{array}
\\
|g(U_n)| \geq n \ \Bigg\Downarrow
&
\Bigg\Downarrow \ \begin{array}{c} B(x, y) := \\ C(f(x), y) \end{array}
\\
\begin{array}{c}
\forall C \ \Pr\big[g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \\
f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big] \in \mathcal{N}
\end{array}
&
\begin{array}{c}
\forall C \ \Pr\big[g\big(C(f \circ g(U_n), 1^n)\big) = \\
g(U_n)\big] \in \mathcal{N}
\end{array}
\end{array}
$$

$$
\Bigg\Downarrow \ \begin{array}{c} \Pr[Q \vee R] = \Pr Q + \Pr R \\ f, g \in \mathcal{N} \ \Rightarrow \ f + g \in \mathcal{N} \\ f \leq g \in \mathcal{N} \ \Rightarrow \ f \in \mathcal{N} \end{array}
$$

$$
\begin{array}{c}
\forall C \ \Pr\big[\big(g\big(C(f \circ g(U_n), 1^n)\big) \neq g(U_n) \ \wedge \\
f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big) \ \vee \\
g\big(C(f \circ g(U_n), 1^n)\big) = g(U_n)\big] \in \mathcal{N}
\end{array}
$$

$$
\Bigg\Downarrow \ \begin{array}{c} (x \neq x' \ \wedge \ f(x) = f(x')) \ \vee \\ x = x' \ \Leftrightarrow \ f(x) = f(x') \end{array}
$$

$$
\forall C \ \Pr\big[f \circ g\big(C(f \circ g(U_n), 1^n)\big) = f \circ g(U_n)\big] \in \mathcal{N} \qquad \square
$$

Conclusion: lesson Some proofs have less clear and redundant parts that make the proofs more difficult to read and longer than they have to be. Proof reorganisation addresses these problems in three steps:

1. writing the proof using some schematic notation that reveals the proof structure;
2. analysing the proof structure looking for
   (a) arguments that can be reordered in a simpler way;
   (b) joint arguments that can be done in a simpler way separately;
   (c) repeated arguments that can be eliminated;
   (d) arguments that are only partially used;

   and so on;
3. rewriting the proof correcting the problems found.

The result is a simpler and/or shorter proof than the original proof.

11.4 Proof presentation: wedding-cake notation

11.7 Case study.

Introduction: idea There are proofs in which part of a formula is repeated unmanipulated through the calculations. This was called "syntactic baggage" and criticised by Dan Grundy (Grundy 2008, page 36). Prompted by this criticism, we propose a notation that aims to avoid "syntactic baggage". This notation has the format exemplified below.

$$\begin{array}{l} \neg(P \;\Rightarrow\; Q) \;\Leftrightarrow\; P \wedge \neg Q \\ \neg(\neg P \vee Q) \\ \neg\neg P \wedge \neg Q \end{array}$$

The selection of a subformula to manipulate while leaving the remaining part of the formula unchanged is reminiscent of Jim Grundy's window inference (Grundy 1993, chapter 2).

Problem: number theory Let us consider the following proposition and its proof.

Proposition. We have $\forall n \in \mathbb{N}\ (n+1)^2(n-1)+(n^3+n^2-n-1) \equiv 0 \pmod 2$. ☐

Proof. Let $n \in \mathbb{N}$. We calculate

$$\begin{aligned} (n+1)^2(n-1) + \underline{(n^3+n^2-n-1)} &= \\ (n+1)((n+1)(n-1)) + \underline{(n^3+n^2-n-1)} &= \\ (n+1)(n^2-1) + \underline{(n^3+n^2-n-1)} &= \\ (n^3+n^2-n-1) + \underline{(n^3+n^2-n-1)} &= \\ 2(n^3+n^2-n-1) &\equiv \\ 0 \pmod 2. \end{aligned}$$ ☐

We can see that almost all formulas occurring in the proof are divided into two parts:

1. a left part that is manipulated through the proof;
2. an underlined right part, a piece of "syntactic baggage", that is not manipulated and just repeated throughout the proof.

The "syntactic baggage" creates three difficulties while writing and reading the proof:

1. we have to keep copying the "syntactic baggage" from one line to another;
2. we have to keep reading the "syntactic baggage" just to realise that it is not manipulated;
3. if we make a mistake in one of the copies of the "syntactic baggage", then the proof becomes incorrect.

Solution: number theory An obvious but uninteresting solution would be to give a name (say α) to the "syntactic baggage" (which is $n^3 + n^2 - n - 1$) and replace the occurrences of the "syntactic baggage" by the name (for example $(n+1)^2(n-1) + (n^3 + n^2 - n - 1)$ would become $(n+1)^2(n-1) + \alpha$). A less obvious but more interesting solution is the notation used in the proof below that allows to manipulate the left parts while avoiding repetitions of the "syntactic baggage".

Proof. Let $n \in \mathbb{N}$. We calculate

$$(n+1)^2(n-1) + (\overbrace{n^3 + n^2 - n - 1}^{\alpha :=}) = 2\alpha \equiv 0 \pmod 2. \qquad \square$$

$$(n+1)((n+1)(n-1))$$

$$(n+1)(n^2 - 1)$$

$$\underbrace{n^3 + n^2 - n - 1}_{=\alpha}$$

We call this notation "wedding cake" notation because when turned upside down it sometimes looks like a wedding cake as shown below.

$$zzz$$
$$yyyyy$$
$$xxxxxxx$$
$$wwwwwwww$$

Problem: cryptography Dan Grundy gave an example (Grundy 2008, page 36) where "syntactic baggage" appears in four out of six lines from a proof in cryptography. We have encountered the same problem: in proof 7.13 (summarised below), the "syntactic baggage" $\Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big)$ (underlined in the summary) appears three times.

Proposition. For all pseudorandom generators G, if G is cryptographically secure, then C_G is indistinguishable from random. $\square$

Proof. The pseudorandom generator G being cryptographically secure means

$$\forall A, A' \;\; \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N}.$$

The stream cipher C_G being indistinguishable from random means

$$\forall B, B' \;\; \underline{\Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big)} - \Pr B'(U'_{|E_G(K(1^n), B(1^n)_1)|}, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Taking $A(x) := (B(x)_1, B(x))$ and $A'(x, y, z) := B'(x \oplus z_1, y, z_2)$ in the premise, we get

$$\forall B, B' \;\; \Pr B'(G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1, 1^n, B(1^n)_2) - \Pr B'(U_{|G(U_n, 1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Substituting $G(U_n, 1^{|B(1^n)_1|}) \oplus B(1^n)_1$ by $E_G(U_n, B(1^n)_1)$, we get

$$\forall B, B' \ \Pr B'\big(E_G(U_n, B(1^n)_1), 1^n, B(1^n)_2\big) - \\ \Pr B'(U_{|G(U_n,1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Substituting U_n by $K(1^n)$, we get

$$\forall B, B' \ \underline{\Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big)} - \\ \Pr B'(U_{|G(K(1^n),1^{|B(1^n)_1|})|} \oplus B(1^n)_1, 1^n, B(1^n)_2) \in \mathcal{N}.$$

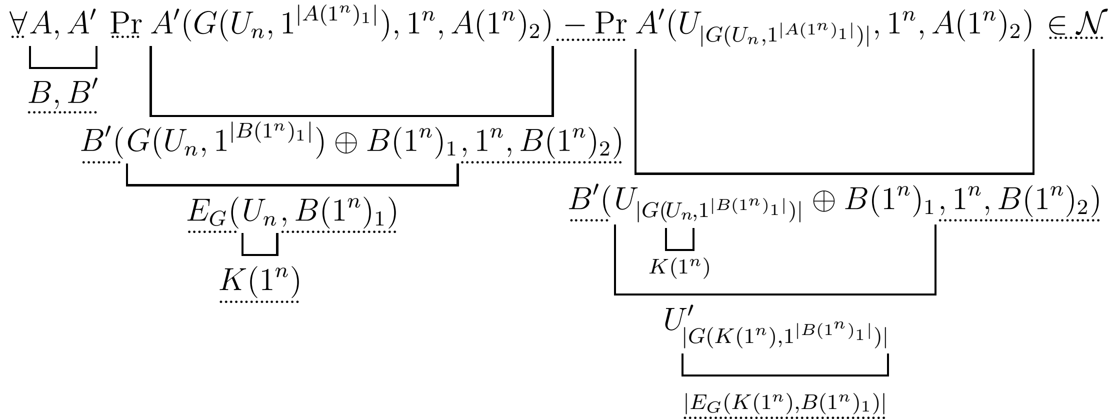
Substituting $U_{|G(K(1^n),1^{|B(1^n)_1|})|} \oplus B(1^n)_1$ by $U'_{|G(K(1^n),1^{|B(1^n)_1|})|}$, we get

$$\forall B, B' \ \underline{\Pr B'\big(E_G(K(1^n), B(1^n)_1), 1^n, B(1^n)_2\big)} - \\ \Pr B'(U'_{|G(K(1^n),1^{|B(1^n)_1|})|}, 1^n, B(1^n)_2) \in \mathcal{N}.$$

Substituting $|G(K(1^n), 1^{|B(1^n)_1|})|$ by $|E_G(K(1^n), B(1^n)_1)|$, we get the conclusion. □

Solution: cryptography As before, we can use the wedding-cake notation that avoids the syntactic baggage: this is done in proof 7.21 (summarised below).

Proof. The top line of the two "wedding cakes" below says that G is cryptographically secure.



The underdotted part says that C_G is indistinguishable from random. □

Conclusion: lesson Dan Grundy's criticism on "syntactic baggage" is justified: there are indeed proofs with "syntactic baggage" that would better be avoided. This led us to the wedding-cake notation. This notation has two advantages:

1. avoids "syntactic baggage";
2. results in a short diagrammatic proof presentation well-suitable as a summary of the full proof.

But it also has two disadvantages:

1. the notation is only easily applied to proofs that are calculational (consisting in manipulating a formula) and linear (having a single line of reasoning from the premise to the conclusion or vice-versa);
2. the notation leaves out justifications of why the manipulations are legal (although the justifications could be added, for example, using a reference system similar to footnotes).

11.5 Proof presentation: carving out a theory

11.8 Case study.

Introduction: idea Sometimes proofs contain implicitly a theory but that theory:

1. does not stand out in isolation because it is mixed with the remaining material of the proof;
2. is not neatly stated in general but less neatly stated only for the particular case used in the proof.

Besides the abstract interest of having the theory, it can be beneficial for the proof presentation:

1. to carve out the theory in the sense of presenting it in isolation and in general;
2. to rewrite the proof as a particular application in context of the general and isolated theory;
3. possibly to use new concepts discovered during the carving out of the theory.

In this case study we are going to see how to carve out a theory from a proof.

Problem: number theory Let us consider the following proposition and its proof.

Proposition. We have $\exists N \in \mathbb{N}\ \forall n \in \mathbb{N}\ (n > N \ \Rightarrow\ n \leq n^3)$. □

Proof. Let $N := 0 \in \mathbb{N}$. Let us take any $n \in \mathbb{N}$. Let us assume $n > N$. Then $1 \leq n$, so $1 = 1 \cdot 1 \leq n \cdot n = n^2$ since 1 and n are non-negative, thus $n = n \cdot 1 \leq n \cdot n^2 = n^3$ since n is non-negative. □

Reading the proof above, we may get the impression that the more general result with $n^i \leq n^{i+j}$ instead of $n \leq n^3$, where $i, j \in \mathbb{N}\backslash\{0\}$, should be provable along the same lines, but the not very neat aspect of the proof does not invite us to try to prove it. Let us try to improve this situation.

Solution: number theory To prove the proposition, and even its generalisation with $n^i \leq n^{i+j}$ instead of $n \leq n^3$, in a neater way, let us construct a "mini-theory" of the "discovered new" concept of eventual domination of functions consisting of a definition and three theorems, and then prove the generalisation of the proposition in a neat way as an application of the "mini-theory".

Definition We say that a function $f\colon \mathbb{N} \to \mathbb{N}$ is *eventually dominated* by a function $g\colon \mathbb{N} \to \mathbb{N}$, and write $f \preceq g$, if and only if

$$\exists N \in \mathbb{N}\ \forall n \in \mathbb{N}\ (n > N \ \Rightarrow\ f(n) \leq g(n)).$$

We often write, for example, $n \preceq n^2$ to mean $f \preceq g$ where the functions $f, g\colon \mathbb{N} \to \mathbb{N}$ are defined by $f(n) := n$ and $g(n) := n^2$.

Theorems For all functions $f, f', g, g', h\colon \mathbb{N} \to \mathbb{N}$, we have

$$1 \preceq n, \tag{11.1}$$
$$f \preceq g \wedge f' \preceq g' \ \Rightarrow\ ff' \preceq gg', \tag{11.2}$$
$$f \preceq g \ \Rightarrow\ fh \preceq gh \tag{11.3}$$

(proof sketch of (11.2): if $n > N \ \Rightarrow\ f(n) \leq g(n)$ and $n > N' \ \Rightarrow\ f'(n) \leq g'(n)$, then $n > \max(N, N') \ \Rightarrow\ f(n)f'(n) \leq g(n)g'(n)$ since $f(n)$, $f'(n)$, $g(n)$ and $g'(n)$ are non-negative).

Application The proposition can be restated in the language of the "mini-theory" as $n \preceq n^3$. We use the "mini-theory" to prove $n \preceq n^3$. In fact, the proof is now so easy that we even prove $\forall i, j \in \mathbb{N} \setminus \{0\}\ n^i \preceq n^{i+j}$ almost without additional effort.

Proof. We have $1 \preceq n$ by (11.1) and we have

$$\begin{array}{lll} & 1 \preceq n \ \Rightarrow & \text{by (11.2) applied } j-1 \text{ times with} \\ & & f(n) = 1 = f'(n) \text{ and } g(n) = n = g'(n) \\ & 1 \preceq n^j \ \Rightarrow & \text{by (11.3) with} \\ & & f(n) = 1,\ g(n) = n^j \text{ and } h(n) = n^i \\ n^i \preceq n^{i+j}, & & \end{array}$$

so we conclude $n^i \preceq n^{i+j}$. $\square$

This new proof, which essentially consists in the three simple steps

$$\stackrel{(11.1)}{\Longrightarrow}\quad 1 \preceq n \quad \stackrel{(11.2)}{\Longrightarrow}\quad 1 \preceq n^j \quad \stackrel{(11.3)}{\Longrightarrow}\quad n^i \preceq n^{i+j},$$

(where $\stackrel{(11.1)}{\Longrightarrow} 1 \preceq n$ means that $1 \preceq n$ is true by (11.1)) not only has a neater aspect than the old proof but even proves a stronger result with almost no additional effort.

Problem: cryptography Proof 8.13 of theorem 8.12 has in it implicit the idea of finding an inverse $x_{|z|}$ of $f(\bar{x})$, where $x_{|z|} \in \{0,1\}^{|z|}$

1. not by searching through all elements of $\{0,1\}^{|z|}$ for $x_{|z|}$ because this is an exponential-time task (since there are $2^{|z|}$ elements);
2. but instead by constructing $x_{|z|}$ bit by bit because this is a polynomial-time task (since it takes $|z|$ steps);

(the proof is complicated so we do not repeat it here, even if summarised). However, this idea is:

1. buried in the middle of other material, which may not make the idea stand out;
2. used in a particular case, which may obscure the general idea.

Solution: cryptography In section 8.4 we gave the neater proof 8.32 of theorem 8.12 obtained by

1. carving out a "mini-theory" about how
   (a) to replace an exponential-time minimisation/search operator μ;
   (b) by a polynomial-time minimisation/search operator $\bar{\mu}$;

   at the expense of potentially increasing the complexity of a P formal language L_f to a NP formal language $\bar{L}_f$;
2. bringing in this way to the foreground the idea of finding an inverse $x_{|x|}$ of $f(\bar{x})$
   (a) not by searching $x_{|x|}$ in $\{0,1\}^{|z|}$ in exponential time using μ;
   (b) but by constructing $x_{|x|}$ bit by bit in polynomial time using $\bar{\mu}$;
3. then applying the "mini-theory" to produce the neater proof 8.32 where essentially $x_{|x|}$ is found in polynomial time by $\bar{\mu}$ as being $\bar{\mu}(\bar{L}_f, f(\bar{x}), 1^{|\bar{x}|})$;
4. using in proof 8.32 the discovered concepts of μ and $\bar{\mu}$.

(again, the proof is complicated so we do not repeat it here, even if summarised).

Conclusion: lesson We saw that carving out a theory from a proof can be beneficial in three ways:

1. it may allow to write a cleaner proof divided into two parts, namely
   (a) the presentation of the theory in isolation and in general;
   (b) the application of the theory in context and in the particular case used in the proof;
2. the theory may even allow to prove a more general result with little additional effort;
3. it has the potential to lead to the discovery of new concepts.

11.6 Proof presentation: direct proof versus indirect proof

11.9 Case study.

Introduction: idea There are two usual methods to prove an implication $P \Rightarrow Q$:

1. the direct proof, in which we assume P and prove Q;
2. the indirect proof (proof by contrapositive), in which we prove $\neg Q \Rightarrow \neg P$ (the contrapositive) by assuming $\neg Q$ and proving $\neg P$.

If $P \;:\Leftrightarrow\; \forall x\ P'(x)$ and $Q \;:\Leftrightarrow\; \forall y\ Q'(y)$ are universal statements, then there is more to be said about the two methods:

1. in the direct proof, we often assume $\forall x\ P'(x)$, take an arbitrary y, instantiate $x = t(y)$ as being some term (or more than one) constructed from y (for example, $x = y^2 + 1$) getting $P'(t(y))$, and prove $P'(t(y)) \;\Rightarrow\; Q'(y)$;
2. in the indirect proof, we often assume $\exists y\ \neg Q'(x)$, take $x = t(y)$ as being some term constructed from the y assumed to exist, and prove $\neg P'(t(y))$.

$$\underbrace{\forall x\ P'(x) \xRightarrow[\text{construction}]{x=t(y)} \forall y\ Q'(y)}_{\text{direct proof}} \qquad \text{versus} \qquad \underbrace{\exists y\ \neg Q'(y) \xRightarrow[\text{construction}]{x=t(y)} \exists x\ \neg P'(x)}_{\text{indirect proof}}$$

In the proofs of this form that we have found in cryptography, often both proof methods work and give proofs that are essentially the same in terms of constructions but different in terms of style. In this case study we are going to examine the advantages and disadvantages of both methods.

Problem: number theory Let us consider the following proposition essentially of the form $\forall x\ P'(x) \;\Rightarrow\; \forall y\ Q'(y)$ and its direct proof.

Proposition. For all functions $f\colon \mathbb{N} \to [0, +\infty[$, if $f \in \mathcal{N}$, then $\sqrt{f} \in \mathcal{N}$. □

Proof. The condition $f \in \mathcal{N}$ means

$$\forall p\ \exists N\ \forall n > N\ |f(n)| < 1/p(n). \tag{11.4}$$

The condition $\sqrt{f} \in \mathcal{N}$ means

$$\forall q\ \exists N\ \forall n > N\ \left|\sqrt{f(n)}\right| < 1/q(n). \tag{11.5}$$

Taking $p := q^2$ in (11.4), we get

$$\forall q\ \exists N\ \forall n > N\ |f(n)| < 1/q(n)^2,$$

so we get (11.5). □

We see two advantages in the above direct proof:

1. direct proofs are often considered more elegant than indirect proofs, which may be a bias but even so plays into the appeal of a proof;
2. direct proofs do not require the law $(\neg Q \;\Rightarrow\; \neg P) \;\Rightarrow\; (P \;\Rightarrow\; Q)$ (nor the laws $\neg\forall x\ P(x) \;\Leftrightarrow\; \exists x\ \neg P(x)$ and $\neg\exists x\ P(x) \;\Leftrightarrow\; \forall x\ \neg P(x)$) of classical logic, so they save some steps (the ones necessary to "expand" the negations of the premise and of the conclusion) and may be formalisable in a weaker logic (such as intuitionistic logic), which has philosophical advantages (some people consider weaker logics more sound) and a constructive advantage (often the so-called proof interpretations to extract constructive/computational content from proofs extract a more meaningful content in weaker logics).

We see, however, also a disadvantage of the direct proof: it is often easier for us to reason along the lines of "from an y such that $\neg Q'(y)$, construct an $x = t(y)$ such that $\neg P'(x)$" than along the lines of "massage $\forall x\ P'(x)$, by constructing $x = t(y)$ from y, into $\forall y\ Q'(y)$", especially when $\neg Q'(y)$ and $\neg P'(x)$ have a clear intuitive meaning (such as often happens in cryptography when they mean the breaking of the security of cryptographic object).

Solution: number theory Now we deal with the disadvantage of direct proofs identified above by giving an indirect proof.

Proof. The condition $\sqrt{f} \notin \mathcal{N}$ means

$$\exists q\ \forall N\ \exists n > N\ \left|\sqrt{f(n)}\right| \geq 1/q(n). \tag{11.6}$$

The condition $f \notin \mathcal{N}$ means

$$\exists p\ \forall N\ \exists n > N\ |f(n)| \geq 1/p(n), \tag{11.7}$$

Taking $p := q^2$, that is $\sqrt{p} = q$, in (11.6), we get

$$\exists p\ \forall N\ \exists n > N\ |\sqrt{f(n)}| \geq 1/\sqrt{p(n)},$$

so we get (11.7). ☐

We see an advantage in the above indirect proof: it follows the easier reasoning "from an y such that $\neg Q'(y)$, construct an $x = t(y)$ such that $\neg P'(x)$" instead of the harder reasoning "massage $\forall x\ P'(x)$, by constructing $x = t(y)$ from y, into $\forall y\ Q'(y)$". We see, however, also two disadvantages in the indirect proof:

1. it is often considered less elegant;
2. requires certain classical laws.

Both direct and indirect proofs have significant advantages and disadvantages, so we do not consider any of them as the "right" proof, instead we consider them as two options available for the prover to choose from according to his/her goal.

We should point out that both proofs above have essentially the same content, which is the construction $p := q^2$. So we would say that the two proofs are essentially the same modulo a change of style between $\forall x\ P'(x) \Rightarrow \forall y\ Q'(y)$ and $\exists y\ \neg Q'(y) \Rightarrow \exists x\ \neg P'(x)$.

Problem: cryptography Let us consider proof 9.10 (summarised below).

Proposition. For all pseudorandom generators G, if G is cryptographically secure, then f_G is one-way. ☐

Proof. The pseudorandom generator G being cryptographically secure means

$$\begin{aligned}\forall A, A'\ \ \Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2) - \\ \Pr A'(U_{|G(U_n,1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2) \in \mathcal{N}.\end{aligned} \tag{11.8}$$

The binary-string function f_G being one-way means (besides being polynomial-time computable)

$$\forall B \ \Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr] \in \mathcal{N}. \tag{11.9}$$

Taking $A(x) := (1^{2|x|}, \epsilon)$ and $A'(x, y, z) := \mathrm{Tr}[f_G(B(x, y)) = x]$ in (11.8), we get

$$\forall B \ \underbrace{\Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr]}_{=:\alpha} - \underbrace{\Pr[f_G(B(U_{2n}, 1^n)) = U_{2n}]}_{=:\beta} \in \mathcal{N}.$$

From $\alpha - \beta \in \mathcal{N}$ above and $\beta \in \mathcal{N}$ because $\beta \leq |f_G^{-1}[\{0,1\}^{2n}]|/|\{0,1\}^{2n}| \leq |\{0,1\}^n|/|\{0,1\}^{2n}| = 2^{-n}$ we get $\alpha \in \mathcal{N}$, so (11.9). $\square$

As before, this proof is direct and so has the advantages of a direct proof but also the disadvantage of the harder reasoning "massage $\forall x \ P'(x)$, by constructing $x = t(y)$ from y, into $\forall y \ Q'(y)$".

Solution: cryptography Now we deal with the disadvantage of the direct proof above giving an indirect proof.

Proof. The pseudorandom generator G *not* being cryptographically secure means

$$\begin{aligned}\exists A, A' \ &\overbrace{\Pr A'(G(U_n, 1^{|A(1^n)_1|}), 1^n, A(1^n)_2)}^{=:\alpha} - \\ &\underbrace{\Pr A'(U_{|G(U_n, 1^{|A(1^n)_1|})|}, 1^n, A(1^n)_2)}_{=:\beta} \notin \mathcal{N}.\end{aligned} \tag{11.10}$$

The binary-string function f_G *not* being one-way means

$$\exists B \ \Pr\bigl[f_G\bigl(B(f_G(U_n), 1^n)\bigr) = f_G(U_n)\bigr] \notin \mathcal{N}. \tag{11.11}$$

Taking $A(x) := (1^{2|x|}, \epsilon)$ and $A'(x, y, z) := \mathrm{Tr}[f_G(B(x, y)) = x]$, where B is given by (11.11), we get

$$\begin{gathered}\alpha \notin \mathcal{N}, \\ \beta = \Pr[f_G(B(U_{2n}), 1^n) = U_{2n}] \in \mathcal{N}\end{gathered}$$

because $\beta \leq |f_G^{-1}[\{0,1\}^{2n}]|/|\{0,1\}^{2n}| \leq |\{0,1\}^n|/|\{0,1\}^{2n}| = 2^{-n}$, so $\alpha - \beta \notin \mathcal{N}$, thus (11.10). $\square$

Again, we should point out that both proofs above have essentially the same content, which is the constructions $A(x) := (1^{2|x|}, \epsilon)$ and $A'(x, y, z) = \mathrm{Tr}[f_G(B(x, y)) = x]$.

Conclusion: lesson Implications of the form $\forall x \ P'(x) \ \Rightarrow \ \forall y \ Q'(y)$ appear often in cryptography. There are two usual methods to prove them:

1. the direct proof reasoning in the form "massage $\forall x \ P'(x)$, by constructing $x = t(y)$ from y, into $\forall y \ Q'(y)$";

2. the indirect proof reasoning in the form "from an y such that $\neg Q'(y)$, construct an $x = t(y)$ such that $\neg P'(x)$".

We made the following comparison between them:

1. the direct proof is often considered more elegant;
2. the direct proof often uses a weaker logic with philosophical and constructive benefits;
3. the indirect proof is often more natural to reason about.

We concluded not to select one them as the "right" proof. Importantly, we noticed that often both proofs are essentially the same in the sense that they contain the same constructions $t(y)$ and differ mostly in style only.

11.7 Proof presentation: quasi-parentheses-free notation

11.10 Case study.

Introduction: idea In formalising statements, sometimes we get long and difficult-to-parse formulas such as the formula

$$\forall X \left(\forall x\, (x \in X \;\Rightarrow\; \exists! y\, P(x,y)) \;\Rightarrow\; \exists Y\, \forall x\, \left(x \in X \;\Rightarrow\; \exists! y\, (y \in Y \wedge P(x,y))\right)\right)$$

expressing the axiom schema of replacement in set theory (saying that if a formula $P(x,y)$ defines on a set X a function f that maps each $x \in X$ to the unique y such that $P(x,y)$, then there exists a set Y that is a codomain of f). The obvious culprit is the proliferation of parentheses. Sometimes we can cut down the number of parentheses by rewriting the formula using standard abbreviations (such as $\forall x \in X \ldots$ for $\forall x\ (x \in X \;\Rightarrow\; \ldots))$ or more ad-hoc rewrites (such as replacing parentheses $(\cdot)$ by square brackets $[\cdot]$ to aid matching pairs of parentheses/brackets), for example as in

$$\forall X\ [\forall x \in X\ \exists! y\ P(x,y) \;\Rightarrow\; \exists Y\ \forall x \in X\ \exists! y \in Y\ P(x,y)],$$

but since the problem occurs often, it seems worthwhile to look for a solution that goes further than saving a few pairs of parentheses. So we explore some quasi-parentheses-free notations in terms of how they perform in two aspects:

1. length reduction measured by:
   (a) the number of symbols;
   (b) the physical length when printed on paper;
2. readability indicated by:
   (a) the absence of parentheses pairs to match;
   (b) the need to introduce white space to improve readability;

3. being unambiguously determined by whether parentheses can be reintroduced in an unique way (modulo redundant parentheses) or not solely by the analysis of the types of the symbols in the formula.

Problem: number theory Let us consider the formula

$$P(0) \;\Rightarrow\; \big(\forall n \in \mathbb{N} \, \big(P(n) \;\Rightarrow\; P(S(n))\big) \;\Rightarrow\; \forall n \in \mathbb{N} \; P(n)\big)$$

expressing the induction axiom for natural numbers, where

1. $P(n)$ is a predicate;
2. $S(n) := n + 1$ is the successor function;

which we take as our fist reference formula for this case study. This formula has four nested level of parentheses because:

1. there are nested implications of the form $A \;\Rightarrow\; ((B \;\Rightarrow\; C) \;\Rightarrow\; D)$;
2. there are nested functions or predicates in $P(S(n))$.

One way to cut down on the number of parentheses is:

1. to rewrite $A \;\Rightarrow\; ((B \;\Rightarrow\; C) \;\Rightarrow\; D)$ as $A \wedge [B \;\Rightarrow\; C] \;\Rightarrow\; D$;
2. to rewrite $P(S(n))$ as $P(n + 1)$;

getting

$$P(0) \wedge \forall n \in \mathbb{N} \, [P(n) \;\Rightarrow\; P(n + 1)] \;\Rightarrow\; \forall n \in \mathbb{N} \; P(n)$$

These rewritings improved this situation but they are ad-hoc and may not work in other situations, so they are not a general solution.

Solution: number theory Below:

1. in (11.12) we repeated the reference formula;
2. in (11.13) we deleted the parentheses (for example $A \;\Rightarrow\; ((B \;\Rightarrow\; C) \;\Rightarrow\; D)$ becomes $A \;\Rightarrow\; B \;\Rightarrow\; C \;\Rightarrow\; D$), creating ambiguity;
3. in (11.14) we added spaces to help parsing the formula (for example $A \;\Rightarrow\; B \;\Rightarrow\; C \;\Rightarrow\; D$ becomes $A \quad\Rightarrow\quad B \Rightarrow C \;\;\Rightarrow\;\; D$), removing the ambiguity;
4. in (11.15) we made the spaces visible (for example $A \quad\Rightarrow\quad B \Rightarrow C \;\;\Rightarrow\;\; D$ becomes $A\text{\textvisiblespace\textvisiblespace} \Rightarrow \text{\textvisiblespace\textvisiblespace}B \Rightarrow C\text{\textvisiblespace} \Rightarrow \text{\textvisiblespace}D$).

$$\begin{aligned}
P(0) \;\Rightarrow\; \big(\forall n \in \mathbb{N} \, \big(P(n) \;\Rightarrow\; P(S(n))\big) \;\Rightarrow\; \forall n \in \mathbb{N} \; P(n)\big), && (11.12)\\
P0 \;\Rightarrow\; \forall n \in \mathbb{N} \; Pn \;\Rightarrow\; PSn \;\;\Rightarrow\; \forall n \in \mathbb{N} \; Pn, && (11.13)\\
P0 \quad\Rightarrow\quad \forall n \in \mathbb{N} \; Pn \Rightarrow PSn \;\Rightarrow\; \forall n \in \mathbb{N} \; Pn, && (11.14)\\
P0\text{\textvisiblespace\textvisiblespace} \Rightarrow \text{\textvisiblespace\textvisiblespace}\forall n \in \mathbb{N} \; Pn \Rightarrow PSn\text{\textvisiblespace} \Rightarrow \text{\textvisiblespace}\forall n \in \mathbb{N} \; Pn. && (11.15)
\end{aligned}$$

The gain of (11.14) relative to (11.12) is:

1. (a) reducing the number of symbols from 34 to 20 (not counting spaces as symbols);
   (b) reducing the length from 301 points to 233 points (a point is $1\,/\,72.27$ inches and we round the points to the unit);
2. (a) reducing the number of levels of parentheses from 4 to 0;
   (b) there was the need to introduce 2 levels of white space (␣ and ␣␣);
3. the formula without parentheses and with white space is unambiguous because we can reintroduce the parentheses in a unique way by:
   (a) writing a first level of parentheses around implications $E \Rightarrow F$ with no spaces;
   (b) writing a second level of parentheses around implications ␣$E \Rightarrow F$␣ with a single space;
   (c) writing a third level of parentheses around implications ␣␣$E \Rightarrow F$␣␣ with two spaces;
   (d) noticing that PSn has two possible meanings, namely $P(S)(n)$ and $P(S(n))$ and excluding $P(S)(n)$ because the predicate P is applied to natural numbers but the function S is not a natural number.

Although there is a gain in removing parentheses, it seems to us that the gain is not enough to justify adopting and getting used to a new notation. Maybe a less drastic solution would be to vary the parentheses: instead of using always round parentheses possibly with different sizes as in $\big(\big(((\cdot))\big)\big)$, we could also use square, curly and angular parentheses as in $\langle\{[(\cdot)]\}\rangle$ and less conventional parentheses such as the quotation marks "'·'", the moustaches $\lmoustache\cdot\rmoustache$ and the doubled parentheses $\langle\!\langle\{\!\{[\![(\!(\cdot)\!)]\!]\}\!\}\rangle\!\rangle$. A less conventional solution would be to replace the range of parentheses by underlines or underbrackets, for example replacing $f(g(h(x)))$ by $f\underline{g\underline{hx}}$ or $f\underbracket{g\underbracket{hx}}$.

Problem: cryptography In formalising statements in cryptography, it is easy to get long and difficult-to-parse formulas such as the formula

$$\begin{gathered}\forall A'\ \exists B\ \forall A, f, g\\ \Pr\big[A'\big(E(K(1^n), A(1^n)_1), f(A(1^n), 1^n), 1^n, A(1^n)_2\big) = g(A(1^n), 1^n)\big] -\\ \Pr\big[B\big(1^{|A(1^n)_1|}, f(A(1^n), 1^n), 1^n, A(1^n)_2\big) = g(A(1^n), 1^n)\big] \in \mathcal{N}\end{gathered}$$

from definition 3.58 (expressing the semantic security of a stream cipher $C = (\mathcal{K}, \mathcal{P}, \mathcal{C}, K, E, D)$), which we take as our second reference formula for this case study.

Solution: cryptography Below:

1. in (11.16) we repeated the reference formula (with the quantifications omitted because they do not have parentheses);
2. in (11.17) we deleted the parentheses (for example $w\big(x(y(z))\big)$ becomes $wxyz$), creating ambiguity;

3. in (11.18) we added spaces to help parsing the formula (for example $wxyz$ becomes $w\quad x\ yz$), removing the ambiguity;
4. in (11.19) we made the spaces visible (for example $w\quad x\ yz$ becomes $w\sqcup\sqcup x\sqcup yz$).

$$\begin{aligned}\Pr\big[A'\big(E(K(1^n),A(1^n)_1),f(A(1^n),1^n),1^n,A(1^n)_2\big)=g(A(1^n),1^n)\big]-\\ \Pr\big[B\big(1^{|A(1^n)_1|},f(A(1^n),1^n),1^n,A(1^n)_2\big)=g(A(1^n),1^n)\big]\in\mathcal{N},\end{aligned}\tag{11.16}$$

$$\begin{aligned}\Pr A'EK1^nA1^n{}_1fA1^n1^n1^nA1^n{}_2=gA1^n1^n-\\ \Pr B1^{|A1^n{}_1|}fA1^n1^n1^nA1^n{}_2=gA1^n1^n\in\mathcal{N},\end{aligned}\tag{11.17}$$

$$\begin{aligned}\Pr\ \ \ A'\ \ E\ K1^n\ A1^n{}_1\ \ f\ A1^n\ 1^n\ \ 1^n\ \ A1^n{}_2\ \ =\ \ g\ A1^n\ 1^n\ \ \ \ -\\ \Pr\ \ \ B\ \ 1^{|A1^n{}_1|}\ \ f\ A1^n\ 1^n\ \ 1^n\ \ A1^n{}_2\ \ =\ \ g\ A1^n\ 1^n\in\mathcal{N},\end{aligned}\tag{11.18}$$

$$\begin{aligned}\Pr\sqcup\sqcup\sqcup A'\sqcup\sqcup E\sqcup K1^n\sqcup A1^n{}_1\sqcup\sqcup f\sqcup A1^n\sqcup 1^n\sqcup\sqcup 1^n\sqcup\sqcup A1^n{}_2\sqcup\sqcup=\sqcup\sqcup g\sqcup A1^n\sqcup 1^n\sqcup\sqcup\sqcup\sqcup-\\ \Pr\sqcup\sqcup\sqcup B\sqcup\sqcup 1^{|A1^n{}_1|}\sqcup\sqcup f\sqcup A1^n\sqcup 1^n\sqcup\sqcup 1^n\sqcup\sqcup A1^n{}_2\sqcup\sqcup=\sqcup\sqcup g\sqcup A1^n\sqcup 1^n\in\mathcal{N}.\end{aligned}\tag{11.19}$$

The gain of (11.18) relative to (11.16) is:

1. (a) reducing the number of symbols from 110 to 63;
   (b) reducing the length from 656 points to 607 points;
2. (a) reducing the number of levels of parentheses from 4 to 0;
   (b) there was the need to introduce 4 levels of white space;
3. the formula without parentheses and with white space is unambiguous.

Although there is a gain, it seems not enough to justify a new notation.

Conclusion: lesson Sometimes there are long and difficult-to-parse formulas. An obvious culprit is the proliferation of parentheses. So we explored quasi-parentheses-free notations. Our conclusion is that although the quasi-parentheses-free notation makes the formulas shorter, it requires white space to help parsing the formulas, and the gain is not enough to justify learning a quasi-parentheses-free notation.

11.8 Conclusion

11.11. In this chapter we

1. recovered some of the previous proof presentations;
2. presented some new proof presentations;

turning each one into a case study with:

1. number-theoretic and cryptographic examples;
2. an extraction of the lesson learned.

Bibliography

Avigad, L. and Goldreich, O. (2011). *Studies in Complexity and Cryptography: Miscellanea on the Interplay between Randomness and Computation*, Vol. 6650 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, chapter Testing Graph Blow-Up, pp. 156–172.

Barthe, G., Grégoire, B., Heraud, S. and Béguelin, S. Z. (2011). Computer-aided security proofs for the working cryptographer, *in* P. Rogaway (ed.), *Advances in Cryptology — Crypto 2011*, Vol. 6841 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 71–90. Proceedings, 31st Annual Cryptology Conference — Crypto 2011, Santa Barbara, California, United States of America, 14–18 August 2011.

Bellare, M., Boldyreva, A. and Palacio, A. (2004). An uninstantiable random-oracle-model scheme for a hybrid-encryption problem, *in* C. Cachin and J. L. Camenisch (eds), *Advances in Cryptology — Eurocrypt 2004*, Vol. 3027 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 171–188. Proceedings, International Conference on the Theory and Applications of Cryptographic Techniques — Eurocrypt 2004, Interlaken, Switzerland, 2–6 May 2004.

Bellare, M., Desai, A., Jokipii, E. and Rogaway, P. (1997). A concrete security treatment of symmetric encryption, *38th Annual Symposium on Foundations of Computer Science*, IEEE [Institute of Electrical and Electronics Engineers] Computer Society, Los Alamitos, California, United States of America, pp. 394–403. Extended abstract. Proceedings, 38th Annual Symposium on Foundations of Computer Science, Miami Beach, Florida, United States of America, 20–22 October 1997.

Bellare, M., Desai, A., Pointcheval, D. and Rogaway, P. (1998). Relations among notions of security for public-key encryption schemes, *in* H. Krawczyk (ed.), *Advances in Cryptology — Crypto '98*, Vol. 1462 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 26–45. Proceedings, 18th Annual International Cryptology Conference — Crypto '98, Santa Barbara, California, United States of America, 23–27 August 1998.

Bellare, M., Hofheinz, D. and Kiltz, E. (2015). Subtleties in the definition of IND-CCA [INDistinguishability under adaptive Chosen-Ciphertext Attacks]: When and how should challenge decryption be disallowed?, *Journal of Cryptology* **28**(1): 29–48.

Bellare, M. and Rogaway, P. (1995). Optimal asymmetric encryption, *in* A. D. Santis (ed.), *Advances in Cryptology — Eurocrypt '94*, Vol. 950 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 92–111. Proceedings, Workshop on the Theory and Application of Cryptographic Techniques — Eurocrypt '94, Perugia, Italy, 9–12 May 1994.

Bhargavan, K., Blanchet, B. and Kobeissi, N. (2017). Verified models and reference implementations for the TLS [Transport Layer Security] 1.3 standard candidate, *2017 IEEE [Institute of Electrical and Electronics Engineers] Symposium on Security and Privacy*, Institute of Electrical and Electronics Engineers, pp. 483–502. Proceedings, 38th 2017 IEEE Symposium on Security and Privacy, San Jose, California, United States of America, 22–24 May 2017.

Blahut, R. E. (2014). *Cryptography and Secure Communication*, Cambridge University Press, Cambridge, United Kingdom.

Blanchet, B. (2016). Automatic verification of security protocols: ProVerif and CryptoVerif, `prosecco.gforge.inria.fr/personal/bblanche/talks/Facebook16.pdf`. Slides. Accessed on 19 September 2018.

Blanchet, B. (2018a). CryptoVerif: Cryptographic protocol verifier in the computational model, `prosecco.gforge.inria.fr/personal/bblanche/cryptoverif`. Webpage. Accessed on 15 September 2018.

Blanchet, B. (2018b). ProVerif: Cryptographic protocol verifier in the formal model, `prosecco.gforge.inria.fr/personal/bblanche/proverif`. Webpage. Accessed on 15 September 2018.

Bleichenbacher, D. (1998). Chosen ciphertext attacks against protocols based on the RSA [Rivest-Shamir-Adleman cipher] encryption standard PKCS [Public Key Cryptography Standards] #1, *in* H. Krawczyk (ed.), *Advances in Cryptology — Crypto '98*, Vol. 1462 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 1–12. Proceedings, 18th Annual International Cryptology Conference — Crypto '98, Santa Barbara, California, United States of America, 23–27 August 1998.

Boiten, E. and Grundy, D. (2010). The logic of large enough, *in* C. Bolduc, J. Desharnais and B. Ktari (eds), *Mathematics of Program Construction*, Vol. 6120 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 42–57. Proceedings, 10th International Conference on Mathematics of Program Construction — MPC 2010, Québec City, Canada, 21–23 June 2010.

Boneh, D. and Boyen, X. (2008). Short signatures without random oracles and the SDH [Strong Diffie-Hellman] assumption in bilinear groups, *Journal of Cryptology* **21**(2): 149–177.

Boneh, D. and Shoup, V. (2017). A graduate course in applied cryptography, `crypto.stanford.edu/~dabo/cryptobook`. Version 0.4. Book draft. Accessed on 8 May 2019.

Boneh, D. and Venkatesan, R. (1998). Breaking RSA [Rivest-Shamir-Adleman cipher] may not be equivalent to factoring, *in* K. Nyberg (ed.), *Advances in Cryptology — Eurocrypt '98*, Vol. 1403 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 59–71. Extended abstract. Proceedings, International Conference on the Theory and Application of Cryptographic Techniques — Eurocrypt '98, Espoo, Finland, 31 May – 4 June 1998.

Brown, D. (1998). *Digital Fortress*, 2009 edn, Corgi Books / Transworld Publishers, London, United Kingdom.

Brown, D. R. L. (2016). Breaking RSA [Rivest-Shamir-Adleman cipher] may be as difficult as factoring, *Journal of Cryptology* **29**(1): 220–241.

Buchmann, J. A. (2001). *Introduction to Cryptography*, Undergraduate Texts in Mathematics, first edn, Springer-Verlag, New York (city), New York (state), United States of America.

Canetti, R. (2009). Introduction to cryptography, `www.cs.tau.ac.il/~canetti/f08-materials`. Lecture notes, course Introduction to Cryptography (0368.4162), Tel Aviv University, Fall of 2008, dated November 2008 – February 2009. Accessed on 26 December 2018.

Comon, H. (2016). Communication security: Formal models and proofs, `www.cs.bris.ac.uk/cryptoschool`. Slides, Summer School on Automatic Verification of Cryptographic Systems / Computer Aided Analysis of Cryptographic Protocols, Universitatea Politehnica din Bucureşti [Politehnica [Technical] University of Bucharest], Bucharest, Romania, 11–14 September 2016. Accessed on 29 September 2016, no longer available on 24 August 2018.

Cremers, C., Dreier, J. and Sasse, R. (2018). Tamarin Prover, `tamarin-prover.github.io`. Webpage. Accessed on 15 September 2018.

Cremers, C., Horvat, M., Hoyland, J., Scott, S. and van der Merwe, T. (2017). A comprehensive symbolic analysis of TLS [Transport Layer Security] 1.3, *CSS'17 — 2017 ACM [Association for Computing Machinery] SIGSAC [Special Interest Group on Security, Audit and Control] Conference on Computer and Communications Security*, Association for Computing Machinery, New York (city), New York (state), United States of America, pp. 1773–1788. Proceedings, 24th 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, Texas, United States of America, 30 October 2017 – 3 November 2017.

Damgård, I. B. (1988). Collision free hash functions and public key signature schemes, *in* D. Chaum and W. L. Price (eds), *Advances in Cryptology — Eurocrypt '87*, Vol. 304 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 203–216. Proceedings, Workshop on the Theory and Application of Cryptographic Techniques — Eurocrypt '87, Amsterdam, Netherlands, 13–15 April 1987.

Damgård, I. B. (2007). A "proof-reading" of some issues in cryptography, *in* L. Arge, C. Cachin, T. Jurdziński and A. Tarlecki (eds), *Automata, Languages and Programming*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, pp. 2–11. Proceedings, 34th International Colloquium on Automata, Languages and Programming 2007, Wrocław, Poland, 9–13 July 2007.

Delfs, H. and Knebl, H. (2007). *Introduction to Cryptography: Principles and Applications*, Information Security and Cryptography, second edn, Springer-Verlag, Berlin, Germany.

Dijkstra, E. W. (2000). The notational conventions I adopt, and why, `cs.utexas.edu/users/EWD/ewd13xx/EWD1300.PDF`. Manuscript, code EWD 1300. Accessed on 5 January 2019.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols, *IEEE [Institute of Electrical and Electronics Engineers] Transactions on Information Theory* **29**(2): 198–208.

Dutle, A. et al. (2009). Problem set 1: Solutions. Problem solutions, course Topics in Discrete Mathematics: Combinatorial Complexity (Math 778C), University of South Carolina, Spring of 2009, dated March 2009. Unpublished, privately communicated, August 2016.

Farshim, P. (2015). Communication between Pooya Farshim and Eerke Boiten.

Galindo, D. (2005). Boneh-Franklin identity based encryption revisited, *in* L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi and M. Yung (eds), *Automata, Languages and Programming*, Vol. 3580 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 791–802. Proceedings, 32nd International Colloquium on Automata, Languages, and Programming — ICALP 2005, Lisbon, Portugal, 11–15 July 2005.

Gaspar, J. (2016). Transformation of cryptographically-secure pseudorandom generators into indistinguishable-from-random stream cipher, *in* J. Brookhouse and G. Parish (eds), *UKSCC 2016: University of Kent School of Computing Conference*, School of Computing, University of Kent, Canterbury, United Kingdom, pp. 17–20. Proceedings, University of Kent School of Computing Conference — UKSCC 2016, School of Computing, University of Kent, Canterbury, United Kingdom, 10 June 2016. Informal publication in internal conference proceedings.

Gaspar, J. and Boiten, E. (2014). Simple composition theorems of one-way functions — proofs and presentations, Cryptology ePrint Archive, Report 2014/1006, `eprint.iacr.org/2014/1006`. Article. Accessed on 24 August 2018.

Gentzen, G. (1935). Untersuchungen über das logische Schließen [Investigations on logical consequence/deduction/inference/reasoning], *Mathematische Zeitschrift [Mathematical Journal]* **39**(2–3): 176–210, 405–431.

Goldreich, O. (2004). *Foundations of Cryptography*, Vol. 1: Basic Tools, digital edn, Cambridge University Press, Cambridge, United Kingdom.

Goldreich, O. (2011). *Foundations of Cryptography*, Vol. 2: Basic Applications, digital edn, Cambridge University Press, Cambridge, United Kingdom.

Goldreich, O., Barak, B., Katz, J., Krawczyk, H. and Koblitz, N. (2007). Letters to the editor, *Notices of the American Mathematical Society* **54**(11): 1454–1456.

Good, J., Michie, D. and Timms, G. (1945). General report on Tunny: With emphasis on statistical methods, *Technical Report HW 25/4–5*, United Kingdom Public Record Office. `www.ellsbury.com/tunny/tunny-000.htm`, `www.alanturing.net/turing_archive/archive/index/tunnyreportindex.html`. Webpages. Accessed on 6 October 2018.

Granville, A. and Martin, G. (2006). Prime number races, *The American Mathematical Monthly* **113**(1): 1–33.

Gries, D. and Schneider, F. B. (1993). *A Logical Approach to Discrete Math*, Texts and Monographs in Computer Science, Springer-Verlag, New York (city), New York (state), United States of America.

Grundy, D. (2008). *Concepts and Calculation in Cryptography*, PhD thesis, University of Kent, Canterbury, United Kingdom.

Grundy, D. and Boiten, E. (2008). Towards a calculational theory of one-way functions. Article. Unpublished.

Grundy, J. (1993). *A Method of Program Refinement*, PhD thesis, University of Cambridge, Cambridge, United Kingdom.

Halevi, S. (2005). A plausible approach to computer-aided cryptographic proofs, Cryptology ePrint Archive, Report 2005/181, `eprint.iacr.org/2005/181`. Article. Accessed on 22 November 2018.

Katz, J. and Lindell, Y. (2015). *Introduction to Modern Cryptography*, Cryptography and Network Security, second edn, Chapman & Hall / CRC [Chemical Rubber Company] Press, Taylor & Francis Group, LLC [Limited Liability Company], Boca Raton, Florida, United States of America.

Klein, A. (2013). *Stream Ciphers*, ebook edn, Springer-Verlag, London, United Kingdom.

Koblitz, N. (2007). The uneasy relationship between mathematics and cryptography, *Notices of the American Mathematical Society* **54**(8): 972–979.

Koblitz, N. and Menezes, A. J. (2006). Another look at "provable security" II, *in* R. Barua and T. Langue (eds), *Progress in Cryptology — Indocrypt 2006*, Vol. 4329 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 148–175. Proceedings, 7th International Conference on Cryptology in India — Indocrypt 2006, Kolkata, India, 11–13 December 2006.

Koblitz, N. and Menezes, A. J. (2007). Another look at “provably security”, *Journal of Cryptology* **20**: 3–37.

Koblitz, N. and Menezes, A. J. (2010). The brave new world of bodacious assumptions in cryptography, *Notices of the American Mathematical Society* **57**(3): 357–365.

Koblitz, N. and Menezes, A. J. (2013). Another look at security definitions, *Advances in Mathematics of Communications* **7**(1): 1–38.

Kohlenbach, U. (2008). *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*, Springer Monographs in Mathematics, first edn, Springer-Verlag.

Lamport, L. (1995). How to write a proof, *The American Mathematical Monthly* **102**(7): 600–608.

Lamport, L. (2012). How to write a 21st century proof, *Journal of Fixed Point Theory and Applications* **11**(1): 43–63.

Menezes, A. (2007). Another look at HMQV [Hashed Menezes-Qu-Vanstone key agreement], *Journal of Mathematical Cryptology* **1**(1): 47–64.

Menezes, A. J., van Oorschot, P. C. and Vanstone, S. A. (1996). *Handbook of Applied Cryptography*, Discrete Mathematics and Its Applications, August 2001 fifth printing edn, CRC [Chemical Rubber Company] Press, Boca Raton, Florida, United States of America.

Mizar Project (2018). Mizar mathematical library, `mizar.uwb.edu.pl/library/`. Webpage. Accessed on 30 September 2018.

Möller, B. (2004). A public-key encryption scheme with pseudo-random ciphertexts, *in* P. Samarati, P. Ryan, D. Gollmann and R. Molva (eds), *Computer Security — ESORICS 2004*, Vol. 3193 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, p. 335–351. Proceedings, 9th European Symposium on Research in Computer Security, Sophia Antipolis, France, 13–15 September 2004.

Nandi, M. (2014). XLS [eXtended by Latin Square] is not a strong pseudorandom permutation, *in* P. Sarkar and T. Iwata (eds), *Advances in Cryptology — ASIACRYPT 2014*, Vol. 8873 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 478–490. Proceedings, Part I, 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, 7–11 December 2014.

Nandi, M. (2015). Revisiting security claims of XLS [eXtended by Latin Square] and COPA [possibly Cipher Online Parallelisable Authenticated], Cryptology ePrint Archive, Report 2015/444, `eprint.iacr.org/2015/444`. Article. Accessed on 3 May 2019.

Papadimitriou, C. H. (1994). *Computational Complexity*, August 1995 reprint edn, Addison-Wesley Publishing Company, Reading, Massachusetts, United States of America.

Pass, R. and Shelat, A. (2010). A course in cryptography, `www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf`. Lecture notes. Accessed on 21 May 2019.

Reingold, O. (1998). *Pseudo-Random Synthesizers, Functions and Permutations*, PhD thesis, The Weizmann Institute of Science, Rehovot, Israel.

Ristenpart, T. and Rogaway, P. (2007). How to enrich the message space of a cipher, *in* A. Biryukov (ed.), *Fast Software Encryption — FSE 2007*, Vol. 4593 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 101–118. Proceedings, Revised Selected Papers, 14th International Workshop, Luxembourg City, Luxembourg, 26-28 March 2007.

Ristenpart, T. and Rogaway, P. (2015). How to enrich the message space of a cipher, Cryptology ePrint Archive, Report 2007/109, `eprint.iacr.org/2007/109`. Article. Accessed on 24 August 2018. Version 20070326:072155 dated March 2007, version 20150227:035315 dated February 2015.

Ristenpart, T., Shacham, H. and Shrimpton, T. (2011). Careful with composition: Limitations of the indifferentiability framework, *in* K. G. Paterson (ed.), *Advances in Cryptology — EUROCRYPT 2011*, Vol. 6632 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 487–506. Proceedings, 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques — EUROCRYPT 2011, Tallinn, Estonia, 15–19 May 2011.

Rogaway, P. (2004). Nonce-based symmetric encryption, *in* B. Roy and W. Meier (eds), *Fast Software Encryption — FSE 2004*, Vol. 3017 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 348–359. Proceedings, 11th International Workshop Fast Software Encryption 2004 — FSE 2004, Delhi, India, 5–7 February 2004.

Rothe, J. (2005). *Complexity Theory and Cryptology: An Introduction to Cryptocomplexity*, Texts in Theoretical Computer Science, Springer-Verlag, Berlin, Germany.

Seeba, I. (2010). CryptoVerif: The tool of crypto analysis, `courses.cs.ut.ee/2010/security-seminar-spring/uploads/Main/seeba-final.pdf`. Article. Accessed on 19 September 2018.

Shannon, C. E. (1949). Communication theory of secrecy systems, *The Bell System Technical Journal* **28**(4): 656–715.

Shoup, V. (2002). OAEP [Optimal Asymmetric Encryption Padding] reconsidered, *Journal of Cryptology* **15**(4): 223–249.

Smart, N. P. (2016). *Cryptography Made Simple*, Information Security and Cryptography, Springer International Publishing, Cham, Switzerland.

Talbot, J. and Welsh, D. (2006). *Complexity and Cryptography: An Introduction*, Cambridge University Press, Cambridge, United Kingdom.

The Tamarin Team (2018). Tamarin-Prover manual: Security protocol analysis in the symbolic model, `tamarin-prover.github.io/manual/tex/tamarin-manual.pdf`. Manual. Accessed on 5 January 2019.

University of Cambridge and Technische Universität München [Technical University of Munich] (1986). Isabelle, `isabelle.in.tum.de`. Webpage. Accessed on 28 September 2018.

Vaudenay, S. (2006). *A Classical Introduction to Cryptography: Applications for Communications Security*, Springer Science+Business Media, Inc., New York (city), New York (state), United States of America.

Velleman, D. J. (2006). *How to Prove It: A Structured Approach*, second edn, Cambridge University Press, Cambridge, United Kingdom.

Wigderson, A. (2008). Letters to the editor, *Notices of the American Mathematical Society* **55**(1): 6–7.