Leakage-Resilient Authentication

MSc thesis
by Joachim H. Schipper

Supervised by Eike Kiltz
and Krzysztof Pietrzak

This thesis was written at CWI (Centrum Wiskunde & Informatica) and submitted to Utrecht University for the degree of Master of Science in Mathematics on April 19, 2010.

Abstract

Leakage-resilient cryptography tries to design algorithms that are provably secure against side-channel attacks, attacks that exploit the physical instead of algorithmic properties of an implementation. We present a message authentication code that is secure in the model proposed by Micali and Reyzin [MR04]; it is essentially a combination of a leakage-resilient pseudorandom generator as found by (Dziembowski and) Pietrzak [Pie09a, DP08] and an ordinary (one-time) MAC. We also give some supplementary results about these generators.

Acknowledgements

Writing this thesis was a lot of fun; however, there were quite a few problems along the road. I am very grateful to my supervisors, Eike Kiltz and Krzysztof Pietrzak, for their suggestions and the pleasant collaboration, and am very happy that I have been given the opportunity to continue this collaboration for at least the coming year. I am also grateful to Frits Beukers, who has agreed to read this work for Utrecht University.

I am thankful that I've been able to attend the workshop "Provable Security against Physical Attacks," held in Leiden, and grateful to the speakers and participants for their insights: I could not have written section 3.1 or section 6.1 without their talks and our private discussions (any errors are, of course, my own.)

Finally, my thanks go to my friends and family. In particular, thanks to Adriaan, Anneke, Grietje and Jacob for their support and enthusiasm. Last but not least, thanks to Esther, for her love, for her patience when I needed (wanted) to do some work, and for the occasional much-needed prod.

Contents

Introduction

Cryptography is everywhere, in the form of wireless internet, cell phones and internet banking. These systems keep our data secure; we even use cryptography to guard our state secrets. In short, modern cryptography is highly successful.

This success is not an accident: wherever possible, cryptographers demand mathematical proofs that the algorithms and protocols work. While exacting, this demand gives us a high degree of confidence in our results; a degree of confidence that would be hard to achieve if we relied only on experience and intuition, as had historically been the case.

Unfortunately, these security proofs almost all rely on the assumption that the device running the cryptographic algorithm is a "black box" — that its inner workings cannot be observed or tampered with. This is, sadly, not always the case in real-world scenarios.

As an example, consider car keys or garage openers. The first generation of these devices just broadcast a password; anyone standing nearby could read it and steal the car while the owner was away.

The car industry responded by switching to a cryptographic scheme. However, the real world is a harsh place for cryptography: for instance, the widely-deployed KeeLoq system was recently defeated [EKM$^+$08]. This attack exploited a so-called *side channel*, a physical property of the device (instead of the mathematical properties of the algorithm considered in the security proofs). Carefully analysing the power used by the device allowed Eisenbarth *et al.* to recover the per-device and manufacturer keys, completely breaking the KeeLoq system (see section 3.1 for an overview.)

KeeLoq was already suspect following a theoretical attack [IKD$^+$08]; additionally, due to a questionable trade-off on the part of the designers, breaking a single KeeLoq device broke all devices in that production run. However, a device without these issues could fall to the same attack.

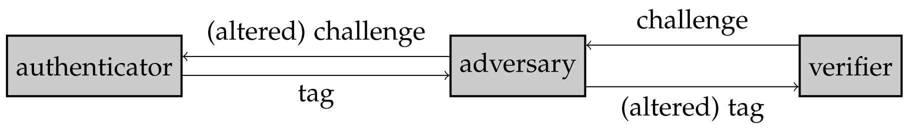


Figure 1: The challenge and response of a MAC-based authentication protocol.

The cryptographic industry (the people actually building KeeLoq, smart cards and other such devices) has worked hard to make side-channel attacks more difficult to carry out. As a consequence, modern devices are quite resistant to attack (KeeLoq was designed before countermeasures against side channels became common.) However, the security of these devices is only safeguarded by experience and intuition, and can usually be broken at "unrealistically high" cost in time, expertise and money: for instance, about a week's work for a well-equipped expert with access to full schematics [Gir10, slide 46].

In contrast, the more theoretically-inclined cryptographers try to deliver provably secure systems which take thousands of years to break, but traditionally does not consider side-channel attacks at all. This has (fairly recently) changed: security proofs have been presented that prove security against (some abstraction of some class of) side-channel attacks. This field is called physical (or leakage-resilient) cryptography.

This thesis presents a *message authentication code* (MAC) that is secure in the theoretical model of leakage proposed by Micali and Reyzin [MR04]. A MAC is a communication protocol which requires that both sides know the same secret ("password.") One side can then calculate a tag, which is a function of both this secret and the message she wishes to send, and send both the message and this "tag." The other side can verify that the tag received matches the message received. (Note that this is *not* encryption: a MAC tries to ensure that a message cannot be changed, whereas encryption tries to ensure that a message cannot be read; see section 2.3.)

The adversary we consider fully controls the communication channel between these devices and can obtain information about the sending device via a side-channel attack (there are some limitations on the information that can be obtained; see section 3.2.) We show that this adversary almost certainly cannot create a valid tag, even for a message chosen by the adversary (of course, this assumes a proper choice of parameters.)

A (leakage-resilient forward-secure) MAC can be used to create a (leakage-resilient) authentication algorithm (and thus, a more secure car key or garage opener.) In an authentication scheme, there are two parties, the *verifier* and the *authenticator*; the authenticator tries to convince the verifier of her identity (see figure 1.) The verifier sends a random challenge and waits for a response (a tag) from the authenticator; she is convinced of the authenticator's identity if she receives a tag such that the pair of challenge and tag is valid.

It is fairly easy to see that the security of the above scheme reduces to the security of the MAC. After all, if the adversary can forge a tag on a random challenge (that she has not seen before,) she can certainly forge a tag on a chosen message; but that would break the MAC.

This thesis is intended to be readable by a motivated student of mathematics; chapter 6 and (the last section of) appendix A may be difficult, though. Those with some knowledge of cryptography should read the next section and then skip to part II; people with knowledge of leakage-resilient cryptography should take note of definition 4.7 (page 22) and start reading either there or in chapter 5.

Notation

We will mostly use standard mathematical notation and fairly standard pseudocode, but we also use a few less common notations.

If S is a set, U_S, U'_S, U''_S and so on are independent random variables distributed according to the uniform distribution on S. If $n \in \mathbb{Z}_{\geq 0}$, U_n, U'_n, U''_n and so on are independent random variables with the same distribution as, but independent from, $U_{\{0,1\}^n}$. If we need more variables, we will use $U_n^{(1)}$, $U_n^{(2)}$ and so forth instead of U_n, U'_n and so forth.

All random variables we consider are discrete. If X is a (discrete) random variable on S, $\Omega_X = \{x \in S \colon P[X = x] > 0\}$. If X and V are random variables, X_v is the random variable defined by $\forall x \in supp(X), v \in \Omega_V \colon P[X_v = x] = P[X = x | V = v]$.

Following Knuth [Knu92], we use the notation $[A]$ for the function which is 1 if the predicate A holds and 0 otherwise ("Iverson's convention"). This was originally introduced for sums and products, e.g. $\sum_{s \in S} s\, [\exists t \in T \colon s = -t]$.

As usual in cryptography, the logarithm implicitly uses base 2 and 0^n resp. 1^n represent the tuples $(0, 0, \ldots, 0)$ resp. $(1, 1, \ldots, 1)$ of length n. In pseudocode, a tuple (x, y) is a value which can be efficiently calculated from x and y and from which x and y can be efficiently calculated.

If x and y are tuples, $x||y$ is the tuple of length $|x| + |y|$ defined by $(x||y)_i = x_i$ for $i \leq |x|$ and $(x||y)_i = y_{i-|x|}$ otherwise. If x is not a tuple, $x||y = (x)||y$; if y is not a tuple, $x||y = x||(y)$. Additionally, $x_{m..n}$ is the tuple $(x_m, x_{m+1}, \ldots, x_{\min(|x|,n)})$ and $x_{[m..n]}$ is the tuple $(x||0^n)_{m..n}$. If $x, y \in \{0,1\}^n$, $x \oplus y$ is the tuple of length n defined by $(x \oplus y)_i = [(x_i, y_i) \in \{(1,0), (0,1)\}]$.

Many cryptographic protocols need a random key, typically generated by a random algorithm `Gen`. We use a random variable K instead, because this is easier and more clear in some places.

Part I

Black-box cryptography

1 Introduction to cryptography

Cryptography has been practised for millennia: famously, Julius Caesar "encrypted" some of his letters by shifting each character three places (so "A" becomes "D," "B" becomes "E," etcetera.) Nonetheless, until the late 20th century, cryptography was badly-understood and practised only by a select few within the military.

Modern cryptography is very different. The algorithms and theory involved are, mostly, publicly available. Modern cryptography is not perfect, but it has replaced the black art of earlier centuries with well-tested explicit assumptions and mathematical derivations.

1.1 Keyed algorithms

Modern cryptography emphasises the use of *keyed algorithms* — that is, schemes that are secure if a small amount of randomly chosen data, the *key*, is unknown to the attacker. Perhaps the biggest advantage of this approach is that the inventor can publish such schemes without rendering them useless (of course, she should not publish the specific key she uses!) If the cryptographic community quickly breaks the proposed scheme, she at least knows not to use it; and if a large number of highly intelligent people have unsuccessfully tried to break the scheme, she can have some confidence that no one else will be able to break the scheme either.

Of course, publishing a scheme has the additional advantage that businesses and individuals can use it, too — which is a net win for a government with a lot of economic value to protect. Additionally, keeping the key secure may be easier than keeping the whole algorithm secure.

In this thesis, we only consider so-called *symmetric* cryptography, i.e. schemes that require that all parties involved share a secret. The alternative is *asymmetric* or *public-key* cryptography; these algorithms can be tremendously useful, but tend to be much slower than their symmetric counterparts.

1.2 Reductions

Clearly, a key goal of cryptography is providing some assurance that a scheme is secure. The gold standard is a reduction in the *standard model*. In the standard model, we assume that an adversary will not be able to observe or tamper with the inner working of the algorithm ("black box.") However, we only limit the computational power of the adversary by specifying e.g. a maximum circuit size and allow her to supply all input and observe all output. We then go on to show that any adversary that can break our security guarantee can be adapted to solve a problem which is assumed to be hard to solve.

With our current knowledge of complexity theory, such a *reduction* is usually the best we can hope for. Being able to *prove* the security of many cryptographic schemes would quickly yield that $\mathbf{P} \neq \mathbf{NP}$. While it is widely believed that $\mathbf{P} \neq \mathbf{NP}$, finding a proof of that statement is the holy grail of complexity theory — in other words, a problem that so far has withstood decades of attempts at solving it.

As an example, the well-known RSA algorithm is related to the problem of factoring large numbers — a problem which, despite millennia of work, still cannot be efficiently solved (see footnote [2] on page 14 for some details on this algorithm.) Of course, factoring is not an unsolvable problem: one can simply try every divisor. However, we need to check a very large number of divisors (about the square root of the value of the key.) Since the RSA algorithm only takes a number of operations related to the *length* of the key, we can choose the keys to be sufficiently long that factoring them is not feasible, but sufficiently short that we are still able to use them reasonably efficiently.

We will work exclusively in this "standard model"; part II will relax the "black box" assumption (see section 3.2.)

1.3 Measuring security

Traditionally, cryptographic statements have been *asymptotic statements*. We gave a somewhat informal example above, where we stated that factoring a number involves "trying a very large number of divisors, about the square root of the value of the key."

More formally, consider an arbitrary polynomial $f\colon \mathbb{Z}_{\geq 1} \to \mathbb{R}$ and an arbitrary series of adversaries $(\mathcal{A}_1, \mathcal{A}_2, \ldots)$ such that $\forall i \in \mathbb{Z}_{\geq 1}\colon |\mathcal{A}_i| \leq f(i)$. (The meaning of $|\mathcal{A}_i|$ is given in section 2.1; if you are more familiar with running time, feel free to substitute that.) Let p_i be the probability that $\mathcal{A}_i$ finds the factorization of an RSA-generated random number in $\{0, 1, 2, \ldots, 2^n - 1\}$. Then there is a polynomial $g\colon \mathbb{Z}_{\geq 1} \to \mathbb{R}$ such that $\forall i \in \mathbb{Z}_{\geq 1}\colon p_i \leq \frac{1}{g(i)}$. Without the formulae, the probability that a *polynomial-size* adversary breaks RSA is *negligible*.

Instead of giving asymptotic statements like the above, we will give exact bounds in our proofs. Such statements are called *concrete statements*; if we state that a scheme is secure in this way, this is called *concrete security*.

2 Some key concepts

We will assume the reader is at least somewhat familiar with complexity theory (Turing machines and (polynomial) running time) — although an informal familiarity will suffice, as this is in no way the focus of this thesis. Additionally, we assume some knowledge of basic probability theory.

All other required concepts from cryptography are introduced succinctly, but some background knowledge will be helpful as we give only minimal details and no proofs. The interested — or confused — reader may wish to refer to a good textbook like Katz and Lindell's *Modern Cryptography* [KL08] and/or Papadimitriou's *Computational Complexity* [Pap94].

2.1 Boolean circuits

(Families of) Boolean circuits are an alternative model of computation to the ubiquitous Turing machines. Circuits may be a better fit for some problems — in particular, circuits are more similar to how a smart card works than Turing machines are. Leakage-resilient cryptography usually uses circuits, for this reason and others. This section owes much to Papadimitriou's textbook [Pap94], although we extend the definition given there. Most importantly, we introduce oracles and allow the output of a circuit to consist of more than one bit.

Properly defining circuits takes a lot of work; however, the basic concept is very simple and can be grasped simply by looking at figure 2.1. With the exception of a few unimportant details, this thesis can probably be completely understood when the following definition is skipped.

Definition 2.1. A *gate* is a function $f\colon \{0,1\}^n \to \{0,1\}^m$, an *input gate* X_i (where $i \in \mathbb{Z}_{\geq 1}$), a *output gate* $\texttt{out}_m$ or a *oracle* $\mathcal{O}_i$. Gates that are functions may be the constant functions 0 or 1, the unary function $\neg\colon \{0,1\} \to \{0,1\}\colon x \mapsto 1 - x$ or the binary functions $\wedge\colon \{0,1\}^2 \to \{0,1\}\colon (x,y) \mapsto xy$ or $\vee\colon \{0,1\}^2 \to \{0,1\}\colon (x,y) \mapsto 1-(1-x)(1-y)$. The *in-degree of a gate* $f\colon \{0,1\}^n$ to $\{0,1\}^m$ is $\deg^{\text{in}}(f) = n$ and the *out-degree* of such a gate is $\deg^{\text{out}}(f) = m$. The in- and out-degree of X_i are 0 and 1; the in- and out-degree of $\texttt{out}_m$ are m and 0; oracles may have any in- and out-degree.

A *(Boolean) circuit* is a tuple $C = (G, A)$, where $G = (X_1, X_2, \dots, X_n, 0, 1, f_1, f_2, \dots, f_p, \mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_q, \texttt{out}_m)$ is a tuple of gates and A is a set of *wires* between these gates, i.e. a set of tuples (i, j, k, l) of positive integers such that $i, k \leq |G|$, $j \leq \deg^{\text{out}}(i)$ and $l \leq \deg^{\text{in}}(k)$. For any such k and l, we require that there are unique $(i^{k,l}, j^{k,l})$ such that $(i^{k,l}, j^{k,l}, k, l) \in A$. Furthermore, A must not have a subset $\{(i_1, j_1, i_2, l_1), (i_2, j_2, i_3, l_2), \ldots, (i_m, j_m, i_1, l_m)\}$.

Let $x \in \{0,1\}^{\deg^{\text{in}}(C)}$. If g_k is a function, $V(C, k, x) = g_k(V(C, i^k_1, x)_{j^k_1}, V(C, i^k_2, x)_{j^k_2}, \ldots, V(C, i^k_{\deg^{\text{in}}(g_k)}, x)_{j^k_{\deg^{\text{in}}(g_k)}})$. Otherwise, if g_k is the special symbol X_i, $V(C, k, x) = x_i$. The *output* of a circuit C without oracles on x is $C(x) = V(C, |G|, x) = (V(C, i^{|G|}_1, x)_{j^{|G|}_1}, V(C, i^{|G|}_2, x)_{j^{|G|}_2}, \ldots, V(C, i^{|G|}_m, x)_{j^{|G|}_m})$. If C contains any oracles, these must be supplied when it is invoked: $C^{\mathsf{o}_1, \mathsf{o}_2, \ldots, \mathsf{o}_q}(x)$. In this case, the oracle gates $\mathcal{O}_i$ are replaced by the function gates o_1 of the same in- and out-degree; the output is then calculated as before.

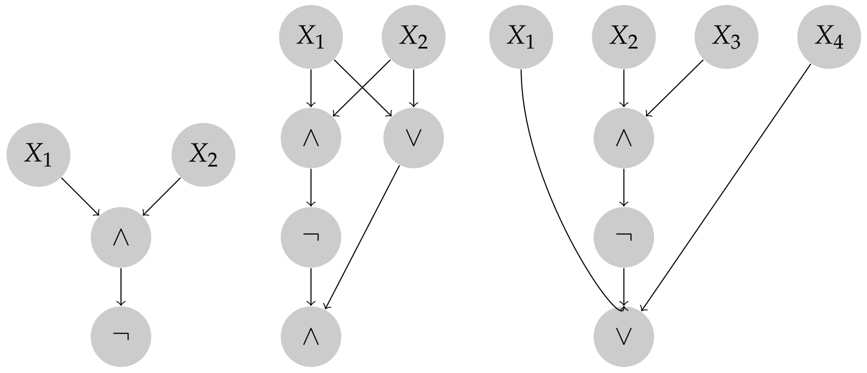
The *in-degree of a circuit* $C = (G, A)$ is the largest integer n such that $G_{1..n} = (X_1, X_2, \ldots, X_n)$. The *out-degree of a circuit* is $\deg^{\text{out}}(C) = \deg^{\text{out}}(G_{|G|}) = \deg^{\text{out}}(\mathtt{out}_m) = m$. The *size* of a circuit is simply the number of gates: $|C| = |G|$. If a circuit calls another circuit and supplies functions to be used as oracles, it must supply one copy of the function for each time the oracle is invoked; the gates implementing these copies are considered part of the calling circuit for purposes of determining the size. (The invoked circuit only needs one gate $\mathcal{O}_i$ for each invocation of the oracle.)

A *family of circuits* C is a set of circuits of distinct in-degrees. Its size is $|C| = \sum_{C_i \in C} |C_i|$. Its output on $x \in \{0,1\}^n$, where the n is such that there is a circuit $C_n \in C$ with $\deg^{\text{in}}(C_n) = n$, is $C(x) = C_n(x)$. We will often simply use "circuit" where a family of circuits is meant. As a technical detail, the function calculated by a family of circuits must be in **PH/poly**.[1]

We do not consider randomized circuits at all: we do not need these to generate our keys, as discussed in chapter , and only consider worst-case adversaries. There is at least one "most lucky draw" for any adversary that uses randomized data that is independent of its input and the results of any oracle queries; replacing the random data by this "most lucky draw" does not decrease the success probability of the adversary. This observation is known as Yao's principle.

It may be interesting to note that the complexity class of polynomial-size circuits includes **BPP**, the class of polynomial-time programs for a Turing machine that give the correct answer with probability at least $\frac{2}{3}$. This follows from an argument similar to Adleman's proof that **R** is in this class [Adl78]. Therefore, even if the above did not hold, the class of polynomial-size circuits includes (almost) all algorithms that are considered practical.

[1]This complexity class is extremely large; the chief use of this restriction is excluding circuits that are themselves small but that can only be constructed after evaluating a ridiculously complex or uncomputable function. This condition is required by Barak *et al.* [BSW03] in a result which is the used in a lemma by Dziembowski and Pietrzak [DP08, lemma 3]. This lemma, in turn, is required for the pseudorandom generators introduced by (Dziembowski and) Pietrzak [DP08, Pie09a], which we use.



(a) X_1 `NAND` X_2 (b) $X_1 \oplus X_2$ (c) $[X = (0,1,1,0)]$

Figure 2.1: Various Boolean functions as circuits

2.2 Pseudorandom generators

Some algorithms require a *lot* of random data (a very large key.) Even worse, many communication protocols require that both sides have access to the *same* random data. It is usually possible to securely generate a small amount of random data: modern operating systems include algorithms to extract randomness from things such as the exact time disk accesses take to complete, or specialized hardware may be used. Furthermore, algorithms like Diffie–Hellman(–Merkle) key exchange usually make it possible to obtain shared random data from private random data. (For more details, see Diffie and Hellman's orginal paper [DH76] or Katz and Lindell's treatment [KL08, section 9.4 and further].)

This leaves us with the problem of generating a *large* amount of random data. It would be nice if we could "stretch" a small amount of random data into a large amount of random data. This is, obviously, impossible — no matter which (deterministic) algorithm is used to turn a randomly-chosen element of $\{0,1\}^{64}$ into an element of $\{0,1\}^{1024}$, the distribution on $\{0,1\}^{1024}$ will be far from uniform. However, we don't really need *random* data — data that "looks sufficiently random" to fool an adversary suffices.

Definition 2.2. Let X, Y and V be random variables, let $s \in \mathbb{Z}$ and let $\varepsilon \in \mathbb{R}$. We say that X is *(ε, s)-indistinguishable from Y* if, for every circuit D of size at most s, $|P[D(X) = 1] - P[D(Y) = 1]| < \varepsilon$. We say that X is *(ε, s)-indistinguishable from Y given V* if (X, V) is (ε, s)-indistinguishable from (Y, V).

This is the main idea behind a class of functions known as *pseudorandom generators* or *stream ciphers*: turn a small amount of random data into a large amount of *pseudorandom* data — data that cannot be distinguished from random.

The name "stream cipher" suggests encryption, and we will not treat encryption in any kind of depth. The interested reader may wish to turn to Katz and Lindell's textbook [KL08, chapter 3].

Definition 2.3. A function $F\colon S \to T$ is a (ε, s)*-secure pseudorandom generator* if $F(U_S)$ is (ε, s)-indistinguishable from U_T.

Of course, only efficient functions F where T is (much) larger than S are used in practice. For more background on pseudorandom generators, see [KL08, section 3.2 and further].

2.3 MESSAGE AUTHENTICATION CODES

Message integrity, along with encryption, is one of the basic problems of cryptography. The problem is most succinctly stated as follows: a message "send € 100 to account 1234" to a bank should not be received as "send € 999 to account 5678," even if the communication channel is insecure. Note that encryption does not suffice to guarantee this; see the end of this section.

Message authentication codes (MACs) are designed to solve the above problem. Essentially, instead of just the message, we send a message plus a "tag," which is computed based on the message and a shared secret. The bank can then verify this tag and discard any messages with a wrong tag.

Definition 2.4. A tuple $\Pi = (K, \mathtt{Tag}, \mathtt{Vrfy})$ is a *message authentication code* if K is a random variable and $\mathtt{Tag}\colon \Omega_K \times \mathbb{M} \to \mathbb{T}$ and $\mathtt{Vrfy}\colon \Omega_K \times \mathbb{M} \times \mathbb{T} \to \{0,1\}$ are circuits. We require that $\mathtt{Vrfy}_k(m, \mathtt{Tag}_k(m)) = 1$ for all $k \in \Omega_K$ and all $m \in \mathbb{M}$.

Of course, for a MAC to be practical, it should be possible to generate K efficiently and `Tag` and `Vrfy` should be fast.

The standard security notion for MACs is *unforgeability*: an adversary cannot produce a valid tag on a message, even when given the ability to request tags on chosen messages. Allowing the adversary to supply arbitrary messages is not unrealistic: in the above example, an attacker could "accidentally" wire someone money and then request it back. This will often cause that person to generate a message with known contents, say "send € 001 to account 5678." Of course, there is a limit to how often this can be done before someone gets suspicious.

```
Algorithm 2.1: $\texttt{BlackBoxForge}^{q}_{\mathcal{A},\Pi}(k)$
    $Q \leftarrow \emptyset$
    $(m,t) \leftarrow \mathcal{A}^{\texttt{TagO}_k(\cdot)}$
    return $[\texttt{Vrfy}_k(m,t) = 1 \wedge |Q| \leq q \wedge m \notin Q]$
```

```
Oracle $\texttt{TagO}_k(m)$
    $Q \leftarrow Q \cup \{m\}$
    return $\texttt{Tag}_k(m)$
```

Definition 2.5. Let $\Pi = (K, \texttt{Tag}, \texttt{Vrfy})$ be a message authentication code. Then Π is (ε, s, q)*-secure in the black-box sense* if, for any circuit $\mathcal{A}$ of size at most s, $P[\texttt{BlackBoxForge}^{q}_{\mathcal{A},\Pi}(K) = 1] < \varepsilon$.

Note that message authentication is different from encryption. On one hand, the message is sent without any attempt at hiding it and can easily be read by an eavesdropper, so MACs do not provide privacy. On the other hand, encryption does not provide message integrity. It is common to encrypt a message m as $m \oplus e$, where e is some data generated by a pseudorandom generator. Clearly, if the message is known, as in the example above, the attacker could send $m \oplus e \oplus m \oplus m'$, where m' is any message of the attacker's choice; decrypting this message will yield m', a message chosen by the attacker.

We will use an equivalent, but slightly different, definition of security for the rest of this thesis: see definition 5.1 and lemma 5.2.

Part II

Leakage-resilient cryptography

3 Introduction to leakage-resilient cryptography

As noted in the introduction, the usual black-box cryptography does not consider side-channel attacks, which unfortunately *are* possible in the real world. Leakage-resilient cryptography strives to model these attacks and then provide algorithms that are provably secure even against these attacks. This would allow us to supplement the rules of thumb and intuition used in industry with some solid mathematical proofs.

To obtain provably secure algorithms, we must (typically) assume that the adversary is bounded in some way. We typically assume a bound on the running time or circuit size; we will also need to assume that the side-channel attacks available to the adversary are somehow limited.

This assumption is not necessarily justified: as a worst-case scenario, consider a computer designed for exhibitions, which shows the contents of its whole memory on the front panel, slowly changing the display as instructions are executed — clearly, hiding secrets from people looking at the front panel is impossible.

More realistically, not all hardware has been designed with security in mind. For instance, basic embedded 8-bit microcontrollers are often highly vulnerable to even "easy" attacks like simple power analysis (see the next section).

Nonetheless, in practice, extracting the internal state from a well-designed device appears to be quite difficult — although typically not impossible.

3.1 Some attacks

Before introducing our model, in the next section, we give an overview of some well-known side-channel attacks, if only to give some flavour. This section does not contain any original material and are required only for a proper understanding of the open questions presented in section 6.1. On the other hand, these attacks do provide the motivation for studying leakage-resilient cryptography in the first place.

Timing attacks. The adversary simply makes the device perform some function and measures how long it takes to do so. Timing attacks are one of the

Algorithm 3.1: $\texttt{Vrfy}_k(m, t)$ (vulnerable implementation)

```
$t' \leftarrow \texttt{Tag}_k(m)$
for $i = 1$ to $|t'|$ {
    if $t_i \neq t'_i$ {
        return 0
    }
}
return 1
```

older and simpler attacks, but can still be effective.

Let $(K, \texttt{Tag}, \texttt{Vrfy})$ be a MAC, and suppose $\texttt{Vrfy}\colon \Omega_K \times \{0,1\}^T \times \mathbb{M} \to \{0,1\}$ is implemented as algorithm 3.1. Note that the number of operations performed by this algorithm is some constant plus a multiple of the number of bits that are equal in t and t'. This allows an attacker to find out that e.g. computing $\texttt{Vrfy}_k(m, 0||0^{T-1})$ takes slightly less time than computing $\texttt{Vrfy}_k(m, 1||0^{T-1})$, computing $\texttt{Vrfy}_k(m, (1,1)||0^{T-2})$ takes slightly less time than computing $\texttt{Vrfy}_k(m, (1,0)||0^{T-2})$, and so on. In this way, a valid tag can be obtained for any message with far fewer operations than it would take to try every possible key.

This issue can be solved by careful implementation: an algorithm that contains no branches (i.e. no **if** statements or the like) is usually impossible to attack in this fashion.[1,2] Nonetheless, algorithm 3.1 was inspired by a real-world vulnerability in a reasonably well-known cryptographic library

[1]Note that merely executing a constant number of instructions across all possible branches — which may be easier to achieve than being completely branch-free — is *not* always sufficient.

Modern CPUs can process data much faster than modern memory can supply data. Fortunately, most programs perform most of their operations on a comparatively small amount of memory. Therefore, modern CPUs include a small amount of very fast (and very expensive) memory, the *CPU cache*. By keeping copies of "interesting" parts of the main memory in the much faster cache, the CPU doesn't have to wait for the main memory quite as much.

This greatly improves performance if the program requests a value from memory that is already in the cache. Unfortunately, this means that code that uses different pieces of memory depending on the input may also be vulnerable to timing attacks, albeit only from attackers that can influence the cache (in practice, that can run programs on the same hardware at the same time.) Percival has described a particularly bad example [Per05].

[2]In particular cases, techniques other than branch-free programming can be simpler and/or faster, e.g. in the case of the RSA algorithm. This algorithm was first described by its inventors Rivest, Shamir and Adleman [RSA78]. We give a succinct one-paragraph introduction below; there are gentler and more complete introductions to this algorithm [RSA78] or [KL08, sections 7.2, 10.1 and 13.3]. Recall that $\varphi(N) = |\{m \in \{1, 2, \ldots, N\}\colon \gcd(N, m) = 1\}|$ and $x^{\varphi(N)} \equiv 1 \mod N$ for all x (Euler's theorem.)

The RSA algorithm requires computing $c^d \mod N$. Here, c is the input, i.e. controlled by the adversary; $N = pq$ and $e \equiv d^{-1} \mod \varphi(N)$ are known to the adversary; d, p, q and $\varphi(N) = (p-1)(q-1)$ are secret, and allowing the adversary to learn any of these breaks the security of the algorithm.

To speed up the computation, $c^d \mod N$ is usually calculated by calculating $c^{d_1} \mod p$ and $c^{d_2} \mod q$ and combining these by using the Chinese Remainder Theorem (d_1 and d_2 are values computed from d for this purpose.) The runtime of many algorithms to calculate $c^{d_2} \mod q$ does, unfortunately, depend on d_2 and/or q. Brumley and Boneh have written a good exposition of an attack based on this property [BB05].

To prevent this attack, we can calculate $(r^e c)^d \cdot r^{\varphi(N)-2} \equiv c^d \mod N$, where r is a random value, instead of computing $c^d \mod N$ directly. The base of the exponentiation is now unpredictable to found by Lawson [Law09].

This is the only side-channel attack presented here that could feasibly be conducted over the internet (as conjectured by Crosby *et al.* [CWR09]; Brumley and Boneh successfully performed such an attack over a switched LAN [BB05, section 5.7].) On the other hand, many attacks in this class requires the ability to run a program on the same hardware as the program being attacked, which may be feasible on some internet-connected servers (which frequently run many programs at different privilege levels) but definitely isn't on smartcards.

Power analysis or EM analysis. Power or EM (electromagnetic field) analysis is conceptually simple: an attacker measures the power consumption of the device, or the electromagnetic field around the device, many times per second. In either case, she can tell how much power is consumed by the device.

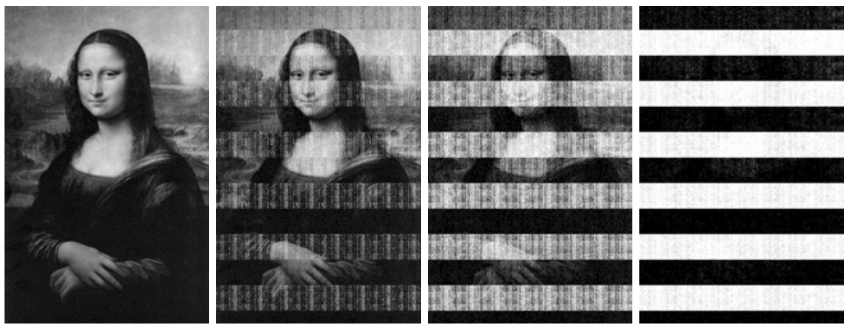
Typically, hardware is designed to be as energy-efficient as possible. This is doubly true of smart cards and other embedded systems where only very limited amounts of power are available. This often means that, while executing a particular task, a piece of hardware only draws enough power to perform that particular task. This is true of the gates in a circuit (which are very similar to the gates in a circuit as defined in section 2.1,) but also for (CMOS) memory: storing data in such memory uses power proportional to the number of *changed* bits.

This attack is mostly a problem for smart cards and other embedded devices — servers are much safer, since they are usually placed in a guarded location, since the circuitry is often extremely complex and since they typically run at a very high speed. That said, power analysis is a very popular and highly effective attack that has broken quite a few real-world systems. Eisenbarth *et al.* describe such an attack on the KeeLoq system [EKM^{+}08].

We have no solid tools to prevent this attack, but all smartcards produced by industry are subjected to such an attack before receiving a certification. This seems to be fairly effective, at least to the extent that breaking such a device really takes as much time and effort as the standard requires. I feel that it is somewhat unfortunate that the highest level of security that can be certified under these standard is still comparatively low, though — the highest level may be given to devices that can be cracked in a week by a well-equipped expert with inside information. There is, perhaps, little danger of a newspaper or competitor breaking such a device, but relying on such a device to keep your data safe against, say, a well-equipped secret service seems foolish.

Cold-boot attacks. These attacks have recently gotten quite a bit of attention. The idea is simple: an attacker simply reads the memory, either after rebooting into a custom operating system or after physically removing the memory from the machine. Modern (DRAM) memory begins "forgetting" soon after the machine loses power; however, this process takes some time (see figure 3.1,) or

the adversary, which defeats such attacks. This countermeasure is called *blinding* and is widely implemented.



(a) After 5 seconds (b) After 30 seconds (c) After 1 minute (d) After 5 minutes

Figure 3.1: A graphical view of (a particular) memory gradually “forgetting” the stored data ([HSH+08, figure 4], used with permission.)

a lot of time if the memory is cooled. During this time, the memory can be read to recover (part of) its contents; from there on, the chief difficulty is finding the encryption keys in the “noisy” memory image and reconstructing them. Halderman *et al.* wrote a very readable article about this attack [HSH+08].

There are other ways of achieving the same result (reading memory.) A common alternative is to attach a device that is given unlimited memory access (DMA access.) This can be done while the computer is running and is much harder to detect than a cold-boot attack. However, such attacks can be defeated simply by making sure that no devices can be attached, while the cold boot attack is harder to guard against. For an example of such an attack, see [Boi06].

These attacks are devastating against laptops with encrypted disks, but harmless if the device has been off for a couple of minutes before an attacker is able to access it. The basic idea — read the memory — is equally viable against smartcards and other dedicated security devices; however, these have typically been engineered to make physically accessing the memory hard, e.g. by encasing the device in an outer layer crisscrossed with wires and erasing the memory as soon as any wire is broken or short-circuited. It probably helps that the certification authorities require a fairly high degree of resistance to such attacks.

Fault induction. Finally, let us consider fault induction — deliberately causing errors to occur in the calculation. This can be done in many ways: focused laser light may scramble part of a circuit, abnormally low or high voltage may cause errors in all sorts of components, or a wire may be connected to a power supply to change the logic of the computation. In many cases, even random faults can be used to easily break the scheme: for instance, Boneh *et al.* show that inducing a single fault in the computations typically used to implement the RSA algorithm suffices to recover the private key [BDL97].

Such attacks are rather hard to execute, and even harder on servers (due to the same issues that make power analysis hard there,) but can be devastating. There are some ways to guard against such attacks — typically by doing the calculation twice or checking the answer by performing the inverse calculation — but we, again, do not have a general theory. Additionally, since these attacks are (correctly) considered very difficult, the certification authorities are not as strict here as elsewhere.

3.2 Modelling side-channel attacks

Following Micali and Reyzin [MR04], we model leakage as a function supplied by the adversary. We restrict this function in two ways: the output must be in a fairly small set and the function is evaluated only on the memory that has been used by this particular calculation. Thus, we assume that previous calculations have no effect on the leakage from the current calculation. We also assume that "only computation leaks information," that is, data that is not accessed is completely secure.

Micali and Reyzin define a variant Turing machine which resembles a conventional computer enough that one can define leakage for such devices. We will take a much simpler approach: we allow the adversary to supply a function each time the device is used (in addition to the input to the device). The supplied function is evaluated on all data that has been touched (read and/or written) by this calculation. The adversary then obtains the output of this function, which must be in a small set, along with the output of the device. We choose this "small set" to be $\{0,1\}^\lambda$ and say that the implementation *leaks* λ *bits.*

In addition to the above, Micali and Reyzin assume that the leakage function is efficient. As a consequence of our definition of circuits, the size of the leakage functions "counts against" the size of the adversary, achieving the same effect.

We introduce a new notation, $x \xleftarrow{k^+} F(k)$, which means "run $F(k)$ and assign its output to x; assign an unambiguous representation of all contents of all memory touched while calculating $F(k)$ to k^+." Note that k^+ includes data that was overwritten as result of evaluating $F(k)$ and the like.

This model is not perfect; see section 6.1 for a discussion. On the other hand, we do not necessarily need the full power of these assumptions; Pietrzak's generator can be proven secure assuming only that certain leakages are uncorrelated [Pie09a, section 1.1], which is weaker than "only computation leaks information."

4 Pseudorandom generators

Pseudorandom generators have been introduced in section 2.2. The relevant security definition (definition 2.3) was, of course, based on a black-box assumption. However, similar constructions are possible in leakage-resilient cryptography.

Unfortunately, we cannot hope for output that is indistinguishable from the uniform distribution. The adversary, after all, can request part of the output as leakage. Therefore, we will first introduce some additional concepts which allow us to express "how random the output is."

4.1 Entropy and pseudoentropy

Consider an adversary guessing the value of a random variable X. If the adversary has no further information, the best guess is the value that X is most likely to take. Note that this probability is 2^{-n} if $X = U_n$ and gets higher as the distribution of X differs more from the uniform distribution. This is the idea behind min-entropy, the most common characterization of "randomness" in cryptography.

Definition 4.1. The *min-entropy* of a random variable X is $H_\infty(X) = -\log \max_{x \in \Omega_X} P[X = x]$.

In many cases, the adversary has some information about X. We can extend the above notion to handle this.

Definition 4.2. Let X and V be random variables. Then the *min-entropy* of X given V is $H_\infty(X|V) = \min_{v \in \Omega_V} H_\infty(X_v)$.

Min-entropy gives a very strong guarantee: *no* adversary — not even one with unlimited resources — can guess the value of a random variable X with probability greater than $2^{-H_\infty(X|V)}$, where V is all the information that the adversary has available.

We introduced pseudorandom generators in section 2.2. These generators are tremendously useful, since schemes using a pseudorandom generator can get by with much less random data than would otherwise be required. However, the output of the pseudorandom generator is not truly random (in the above terms, the min-entropy of the output is at best as large as the min-entropy of the input), but does "look random" to adversaries with limited resources. We can formalize this using HILL-pseudoentropy.

Definition 4.3 ([HILL99, HLR07, DP08][1]). Let X, Y and V be random variables, let $s \in \mathbb{Z}$ and let $\varepsilon \in \mathbb{R}$. The (ε, s)*-HILL-pseudoentropy* of X is at least p, denoted as $H^{\mathrm{HILL}}_{\varepsilon,s}(X) \geq p$, if X is (ε, s)-indistinguishable from Y and $H_\infty(Y) \geq p$. The (ε, s)*-HILL-pseudoentropy of X given V* is at least p, denoted as $H^{\mathrm{HILL}}_{\varepsilon,s}(X|V) \geq p$, if X is (ε, s)-indistinguishable from Y given V and $H_\infty(Y|V) \geq p$.

Note that the HILL-pseudoentropy is, indeed, at least as high as the min-entropy. Consider random variables X and V. Then $H^{\mathrm{HILL}}_{\varepsilon,s}(X|V) \geq H_\infty(X|V)$ for all $s \in \mathbb{Z}$ and $\varepsilon \in \mathbb{R}$, since X is clearly (ε, s)-indistinguishable from X itself (given V.)

It may be instructive to consider the pseudoentropy of U_n given $[U_n = 0^n]$. Note that $[U_n = 0^n]$ is almost never 1; on the other hand, if it is, guessing U_n becomes *very* easy. We could naively say that $H^{\mathrm{HILL}}_{0,s}(U_n|\,[U_n = 0^n]) = 0$, but that is not very useful. However, U_n is indistinguishable from $U_{\{0,1\}^n \setminus \{0^n\}}$ exactly if $U_n \neq 0^n$. Therefore, $H^{\mathrm{HILL}}_{2^{-n},s}(U_n|\,[U_n = 0^n]) \geq \log(2^n - 1)$.

Let us now consider the probability that an adversary guesses correctly. If $U_n = 0^n$, all bets are off; otherwise, the probability of guessing correctly is $\frac{1}{2^n-1}$, for a maximum total success probability of $\frac{1}{2^n} \cdot 1 + \frac{2^n-1}{2^n} \cdot \frac{1}{2^n-1} = \frac{1}{2^{n-1}}$. This make sense, given that the adversary essentially gets to guess twice (0^n and a value of her choice.)

As can be seen from the citations above, HILL-pseudoentropy is not a new concept. Nonetheless, it can be difficult to handle. For instance, if X is a random variable and $f\colon \Omega_X \to \{0,1\}^\lambda$ is any function, then $H_\infty(X|f(X)) \approx H_\infty(X) - \lambda$ with high probability (we obviously require that $\lambda \leq |\Omega_X|$; see proposition A.1.) If f can be calculated by a circuit, we'd expect that, likewise, $H^{\mathrm{HILL}}_{\varepsilon,s}(X|f(X)) \approx H^{\mathrm{HILL}}_{2^\lambda\varepsilon,s-|f|}(X) - \lambda$. We conjecture that this is not true; the interested reader can find our reasoning in section A.1.

4.2 LEAKAGE-RESILIENT PSEUDORANDOM GENERATORS

Pseudorandom generators are as useful in leakage-resilient cryptography as in black-box cryptography. We need significantly more complex algorithms and security proofs, though.

Consider a generator, incrementally generating "chunks" of pseudorandom data from a single, unchanging key. This scheme is trivially insecure in our model of leakage: the adversary can simply request the first λ bits of the key

[1]The concept of HILL-pseudoentropy was first introduced (with some differences from the modern definition) by Håstad *et al.* in [HILL99]. Barak *et al.* have written a good overview [HLR07], which also includes a lot of material on the relationship between HILL-pseudoentropy and some related notions. Conditional min-entropy was used by Dziembowski and Pietrzak [DP08].

as leakage from the first round, the next λ bits of the key as leakage from the second round, and so on. Clearly, this would allow the adversary to learn the entire key, breaking our scheme. Therefore, the generator has to be stateful, which allows us to change the key during operation.

Definition 4.4. A tuple $(K, \mathtt{S})$ is a *stateful pseudorandom generator* if $\mathtt{S}\colon \mathbb{K} \to \mathbb{K} \times \mathbb{X}$ is a circuit and K is a random variable on $\mathbb{K}$.

For any $k \in \mathbb{K}$, we define $k_0^{\mathtt{S}}(k) = k$ and, for $i \in \{1, 2, \ldots\}$, $(k_i^{\mathtt{S}}(k), X_i(k)) = \mathtt{S}(k_{i-1}^{\mathtt{S}})$. Where this does not cause confusion, we will simply write k_0, k_1, X_1, k_2, X_2 and so forth.

The above is not a new idea: as alluded to in section 2.2, almost all "real" pseudorandom generators are defined in this way. Such a construction has many advantages; most obviously, it allows a programmer to request "random" data as needed.

Unfortunately, a naive stateful generator is no more secure than its stateless counterpart. Consider an algorithm that generates a single key, and, each round, uses the entire key to generate output and a new key. No matter the exact algorithms used, this is insecure: future keys can be calculated efficiently from the current key, and therefore the adversary can request part of the future key as leakage from the current round. (As noted in section 6.1, one could argue that this "precalculation attack" is an artefact of our model rather than a realistic threat; but we do not have a model that is clearly better.) Repeating these requests as above would then disclose some future key in its entirety.

Under the assumption that "only computation leaks information," this problem can be solved. To get around it, the stateful pseudorandom generators introduced by (Dziembowski and) Pietrzak [DP08, Pie09a] have two independent keys and use these keys in alternating rounds.

Of course, we must precisely define what it means for such a generator to be secure. We will give several definitions. Let us first define an analogue of the security notion given in definition 2.3.

Definition 4.5. A stateful pseudorandom generator $(K, \mathtt{S})$, as above, is (ε, s, q)-*secure* if $(X_1, X_2, \ldots, X_q)$ is (ε, s)-indistinguishable from $(U_{\mathbb{X}}^{(1)}, U_{\mathbb{X}}^{(2)}, \ldots, U_{\mathbb{X}}^{(q)})$.

The above definition is simple and perfectly fine, but does not handle leakage at all. The following definition is the main security notion used by (Dziembowski and) Pietrzak [DP08, Pie09a]; their generators are secure in this sense.

Algorithm 4.1: $\mathtt{Observe}^{\lambda}_{\mathcal{A},\mathtt{S}}(k)$

```
$\Lambda \leftarrow 0^0$
$\mathcal{A}^{\mathtt{SO}^{\lambda}_{k}(\cdot)}()$
**return** $\Lambda$
```

Oracle $\mathtt{SO}^{\lambda}_{k}(f)$

```
$(k, X) \xleftarrow{k^+} \mathtt{S}(k)$
$\Lambda \leftarrow \Lambda || f(k^+)_{[1..\lambda]}$
**return** $(X, f(k^+)_{[1..\lambda]})$
```

Definition 4.6 ([DP08, Pie09a][2]**).** A stateful pseudorandom generator $(K, \mathtt{S})$, where $\mathtt{S}\colon \mathbb{K} \to \mathbb{K} \times \mathbb{X}$, is *$\lambda$-past-leakage resilient (ε, s)-pseudorandom for q queries* if X_q is (ε, s)-indistinguishable from $U_{\mathbb{X}}$ given $(X_1, X_2, \ldots, X_{q-1}, \Lambda_{[1..(q-1)\lambda]})$, where $\Lambda = \mathtt{Observe}^{\lambda}_{\mathcal{A},\mathtt{S}}(K)$.

Essentially, the above definition guarantees an attacker cannot recover the key from the output and leakage of the generator, nor can she in any other way obtain future output; however, it does not tell us anything about the quality of the output while the device is leaking. The following definition is much more powerful and much better suited to our purposes; we will exclusively consider this definition in chapter 5.

Definition 4.7. A stateful pseudorandom generator $(K, \mathtt{S})$, where $\mathtt{S}\colon \mathbb{K} \to \mathbb{K} \times \mathbb{X}$, is *$\lambda$-leakage resilient (ε, s, α)-forward secure for q queries* if, for each circuit $\mathcal{A}$, $H^{\mathrm{HILL}}_{\varepsilon, s-|\mathcal{A}|}(X_q | X_1, X_2, \ldots, X_{q-1}, \Lambda_{[1..(q-1)\lambda]}, k_q) \geq \mathbb{X} - \alpha$, where $\Lambda = \mathtt{Observe}^{\lambda}_{\mathcal{A},\mathtt{S}}(K)$.

This definition is different from definition 4.6 in several ways. Unlike that definition, it does guarantee that the output is hard to predict even while the device is leaking. We pay for this extra power by the fact that the output is no longer indistinguishable from uniform but merely from something with high min-entropy.

This new security notion also includes *forward security*: if an adversary does obtain the key, she cannot calculate the past output. Depending on the protocol, this may severely limit the damage that can be done. For instance, when using a pseudorandom generator to encrypt data, it means that past messages cannot be read when the key is compromised. Indeed, 0-leakage resilient $(\varepsilon, s, 0)$-forward security for q queries is just the usual (black box) notion of forward security for pseudorandom generators; see, for instance, [BY03, section 2.2] (the rest of that paper is a good overview of forward security in the context of pseudorandom generators and MACs, and well worth a read too.)

[2]These definitions are somewhat scattered. Read at least "Indistinguishability" in section 2 and theorem 1 in the first article and "Leakage-Resilient Security Notion", again in section 2, and theorem 2 in the latter article.

Note that (Dziembowski and) Pietrzak prove that their generators are past-leakage resilient secure, as can be seen from [DP08, theorem 1] and [Pie09a, theorem 2]. However, we can prove that a slight variant of Pietrzak's generator (and Dziembowski and Pietrzak's generator) is also secure in the sense of definition 4.7. For the gory technical details, see section A.3.

Note that this is not true in general: a generator may be past-leakage resilient pseudorandom without being leakage-resilient forward secure. Consider any generator that is secure in the first sense with λ bits of leakage and produces $X_i \in \{0,1\}^n$ for some n. Now consider the generator that just runs this generator and then outputs $(X_i)_{1..\lambda}$. It is no easier to distinguish between $(X_i)_{1..\lambda}$ and $(Y_i)_{1..\lambda}$ than it was to distinguish between X_i and Y_i (in both cases, given past output and leakage; Y_i is as in definition 4.3,) so this new generator is still λ-past-leakage resilient pseudorandom; but it's clearly not leakage-resilient forward secure, since the adversary can just request $(X_i)_{1..\lambda}$ as leakage.

There are many more security notions that make sense, for instance (Dziembowski and) Pietrzak's past-leakage resilient forward security [DP08, Pie09a]. The interested reader is referred to section A.1 and section A.2, where we conjecture that their proof does not work and provide an alternative to the step which may be wrong.

There are more fundamental open questions, too; see section 6.2.

5 Message authentication codes

As in the previous chapter, the security notion we introduced for MACs (definition 2.5) is based on a black-box assumption. We provide a security notion that is useful in leakage-resilient cryptography and a construction that achieves this level of security.

Of course, merely having a definition is not particularly useful. We start by reformulating definition 2.5 in such a way that our security notion for leakage-resilient cryptography is clearly an extension; this is section 5.1. We go on to prove a technical result about MACs (section 5.2) before giving our definitions (section 5.3.) Finally, we give a construction and prove it secure in section 5.4, solving the main problem as presented in the introduction.

5.1 Security in the standard setting

We have defined a security notion for MACs in section 2.3 (definition 2.5). In this chapter we define a security notion that does consider leakage. The following definition forms the "missing link"; we show it's equivalent to definition 2.5 and it's also obviously related to definition 5.6.

```
Algorithm 5.1: $\texttt{Forge}^{q}_{\mathcal{A},\Pi}(k)$
    $Q \leftarrow \emptyset$
    $r \leftarrow 0$
    $\mathcal{A}^{\texttt{TagO}_k(\cdot),\texttt{VrfyO}^{q}_{k}(\cdot,\cdot)}$
    return $r$
```

```
Oracle $\texttt{TagO}_k(m)$
    $Q \leftarrow Q \cup \{m\}$
    return $\texttt{Tag}_k(m)$
```

```
Oracle $\texttt{VrfyO}^{q}_{k}(m,t)$
    $Q \leftarrow Q \cup \{(m,t)\}$
    $v \leftarrow \texttt{Vrfy}_k(m,t)$
    if $v = 1 \wedge m \notin Q \wedge |Q| < q$ {
        $r \leftarrow 1$
    }
    return $v$
```

Definition 5.1. Let $\Pi = (K, \texttt{Tag}, \texttt{Vrfy})$ be a message authentication code. Then Π is (ε, s, q)-*secure* if $P[\texttt{Forge}^q_{\mathcal{A},\Pi}(K) = 1] < \varepsilon$ for any circuit $\mathcal{A}$ of size at most s.

Lemma 5.2. *Let Π be a message authentication code. If Π is (ε, s, q)-secure in the above sense, it is $(\varepsilon, s-1, q-1)$-secure in the sense of definition 2.5. If Π is $(\varepsilon, s, q-1)$-secure in that sense, it is $(q\varepsilon, s, q)$-secure in the above sense.*

Proof. Suppose Π is (ε, s, q)-secure in the above sense. Let $\mathcal{A}$ be any circuit of size at most $s-1$; let $\mathcal{A}'(k) = \texttt{VrfyO}_k(\mathcal{A}(k))$. Note $|\mathcal{A}'| = |\mathcal{A}| + 1 \leq s$, and $P[\texttt{BlackBoxForge}^{q-1}_{\mathcal{A},\Pi}(K) = 1] \leq P[\texttt{Forge}^q_{\mathcal{A}',\Pi}(K) = 1] \leq \varepsilon$.

Suppose Π is $(\varepsilon, s, q-1)$-secure in the sense of definition 2.5. Let $\mathcal{A}$ be any circuit of size at most s and let $i = \operatorname{argmax}_{i \in \{1,2,\ldots,q\}} P[\texttt{Forge}^i_{\mathcal{A},\Pi}(K) = 1] - P[\texttt{Forge}^{i-1}_{\mathcal{A},\Pi}(K) = 1]$. (That is, let i be the number of a round in which the adversary is most likely to first forge a message.) Now define $\mathcal{A}'$ as a circuit which performs the same operations as $\mathcal{A}$, except that all queries to $\texttt{VrfyO}$ are replaced by the constant 0, except if it is the i-th query, in which case $\texttt{VrfyO}_k(m,t)$ is replaced by $\textbf{return}(m,t)$. Then $|\mathcal{A}'| \leq |\mathcal{A}| \leq s$ and

$$\begin{aligned}
&P[\texttt{Forge}^q_{\mathcal{A},\Pi} = 1] \\
&\quad \leq \sum_{i=1}^{q} P[\texttt{Forge}^i_{\mathcal{A},\Pi}(K) = 1] \\
&\quad \leq q(P[\texttt{Forge}^i_{\mathcal{A},\Pi}(K) = 1] - P[\texttt{Forge}^{i-1}_{\mathcal{A},\Pi}(K) = 1]) + \\
&\qquad P[\texttt{Forge}^0_{\mathcal{A},\Pi}(K) = 1] \\
&\quad = qP[\texttt{BlackBoxForge}^{q-1}_{\mathcal{A}',\Pi}(K) = 1] + 0 \\
&\quad < q\varepsilon
\end{aligned}$$

□

5.2 MACs with weak keys

In practice, almost all MACs generate a key from a uniform distribution. However, we are interested in leakage-resilient cryptography, and we cannot obtain uniform randomness in that setting. Therefore, it is natural to consider MACs for "weak keys," that is, keys that are less than perfectly random.

Theorem 5.3. *Let $\Pi = (K, \texttt{Tag}, \texttt{Vrfy})$ be an (ε, s, q)-secure MAC. Let K' be a random variable such that $\forall k \in \Omega_{K'}: P[K' = k] \leq \Delta P[K = k]$. Then $(K', \texttt{Tag}, \texttt{Vrfy})$ is a $(\Delta\varepsilon, s, q)$-secure MAC.*

Proof. Let $\mathcal{A}$ be any circuit of size at most s. Then

$$\begin{aligned}
P[\texttt{Forge}^q_{\mathcal{A},\Pi}(K') = 1] &= P[K' \in \{k : \texttt{Forge}^q_{\mathcal{A},\Pi}(k) = 1\}] \\
&= \sum_{k \in \{k : \texttt{Forge}^q_{\mathcal{A},\Pi}(k) = 1\}} P[K' = k]
\end{aligned}$$

$$\begin{aligned}
&\leq \sum_{k \in \{k : \mathtt{Forge}^{q}_{\mathcal{A},\Pi}(k)=1\}} \Delta P[K = k] \\
&= \Delta P[\mathtt{Forge}^{q}_{\mathcal{A},\Pi}(K) = 1] \\
&\leq \Delta\varepsilon
\end{aligned} \qquad \square$$

While the above is very general, its condition is somewhat uncommon in cryptographic theorems (in fact, it's the concept of "density" as used by Reingold *et al.* [RTTV08].) The following formulation will look more familiar.

Corollary 5.4. *Let $\Pi = (U_{\mathbb{K}}, \mathtt{Tag}, \mathtt{Vrfy})$ be an (ε, s, q)-secure MAC. Let K' be a random variable on $\mathbb{K}$ with min-entropy $\log(|\mathbb{K}|) - \alpha$. Then $(K', \mathtt{Tag}, \mathtt{Vrfy})$ is a $(2^{\alpha}\varepsilon, s, q)$-secure MAC.*

Proof. For any $k \in \mathbb{K}$, $P[K' = k] \leq 2^{-H_{\infty}(K)} = 2^{\alpha} P[U_{\mathbb{K}} = k]$. Now apply theorem 5.3. $\square$

5.3 Stateful MACs

Just as a stateless pseudorandom generator cannot be secure when leakage is considered (see section 4.2,) a "standard" MAC cannot be secure either: an adversary can very easily obtain the key used. Therefore, let us define stateful MACs.

Algorithm 5.4: $\mathtt{SelfCheck}_{\Pi}(k, k', M, T)$

```
for $m \in M_{1..|M|-1}$ {
    $k \leftarrow \mathtt{Tag}_k(m)_1$
}
for $(m, t) \in T$ {
    $k' \leftarrow \mathtt{Vrfy}_{k'}(m, t)_1$
}
$k \leftarrow \mathtt{Tag}_k(M_{|M|})_2$
return $\mathtt{Vrfy}_{k'}(M_{|M|}, t)_2$
```

Definition 5.5. A tuple $\Pi = (K, K', \mathtt{Tag}, \mathtt{Vrfy})$ is a *stateful message authentication code* if K and K' are random variables and $\mathtt{Tag}\colon \Omega_K \times \mathbb{M} \to \Omega_K \times \mathbb{T}$ and $\mathtt{Vrfy}\colon \Omega_{K'} \times \mathbb{M} \to \Omega_{K'} \times \{0,1\}$ are circuits. We require that $\forall k \in \Omega_K, k' \in \Omega_{K'}, n \in \mathbb{Z}_{\geq 0}, M \in \mathbb{M}^n, T \in (\mathbb{M} \times \mathbb{T})^{n+1}$: $\mathtt{SelfCheck}_{\Pi}(k, k', M, T) = 1$.

Note that a "stateless" MAC as defined in definition 2.4 is just a stateful MAC that never updates its key; in that case, $\mathtt{SelfCheck}_{\Pi}(k, k', M, T)$ is equivalent to the correctness requirement given in that definition.

The following security notion is useful in the context of leakage-resilient cryptography. We first allow the adversary to query both `TagO` and `VrfyO` (since the key evolves, being able to query `VrfyO` as well as `TagO` makes the adversary slightly more powerful; it's somewhat surprising that Bellare and Yee [BY03, section 3.2] do not mention this.) We subsequently reveal the key used by TagO. The adversary wins if she can generate a valid tag on a message that has not been seen by TagO, either before or after receiving the key. In the latter case, only tags which were generated using an "earlier" key count.

Note the similarity between this definition and definition 5.1. As in section 4.2, splitting the adversary into two parts does not constitute a significant limitation.

Algorithm 5.5: $\texttt{Forge}^{q,\lambda}_{\mathcal{A},\Pi}(k,k')$

```
$Q \leftarrow \emptyset$
$r \leftarrow 0$
$i_{\texttt{Tag}} \leftarrow 0$
$i_{\texttt{Vrfy}} \leftarrow 0$
$\mathcal{A}_1^{\texttt{TagO}^{\lambda}_{k}(\cdot),\texttt{VrfyO}^{q}_{k'}(\cdot,\cdot)}()$
$\mathcal{A}_2^{\texttt{VrfyO}^{\min(i_{\texttt{Tag}}-1,q)}_{k'}(\cdot,\cdot)}(Q,k)$
return $r$
```

Oracle $\texttt{TagO}^{\lambda}_{k}\,(m,f)$

```
$i_{\texttt{Tag}} \leftarrow i_{\texttt{Tag}} + 1$
$(k,t) \overset{k^+}{\longleftarrow} \texttt{Tag}_k(m)$
$Q \leftarrow Q \cup \{(i_{\texttt{Tag}}, m, t, f, f(k^+)_{[1..\lambda]})\}$
return $(t, f(k^+)_{[1..\lambda]})$
```

Oracle $\texttt{VrfyO}^{q}_{k'}\,(m,t)$

```
$i_{\texttt{Vrfy}} \leftarrow i_{\texttt{Vrfy}} + 1$
$(k',v) \overset{k'^+}{\longleftarrow} \texttt{Vrfy}_{k'}(m,t)$
if $v = 1 \wedge (i_{\texttt{Vrfy}}, m, t) \notin \{q'_{1..3} \colon q' \in Q\} \wedge |Q| \leq q$ {
    $r \leftarrow 1$
}
$Q \leftarrow Q \cup \{(i_{\texttt{Vrfy}}, m, t, v)\}$
return $v$
```

Definition 5.6. Let $\Pi = (K, \texttt{Tag}, \texttt{Vrfy})$ be a stateful message authentication code. Then Π is *λ-leakage resilient (ε, s, q)-forward secure* if, for any tuple of circuits $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $|\mathcal{A}_1| + |\mathcal{A}_2| \leq s$, $P[\texttt{Forge}^{q,\lambda}_{\mathcal{A},\Pi}(K,K') = 1] < \varepsilon$.

5.4 MAIN CONSTRUCTION

We will now propose a scheme that satisfies the security definition given above (definition 5.6.) The basic idea is simple: we take a leakage-resilient pseudorandom generator and use it to generate keys for a (stateless) MAC. We know that the output of the pseudorandom generator has high pseudoentropy and can use theorem 5.3 to prove that the resulting combination is secure.

Algorithm 5.8: $\mathtt{Tag}^{\mathtt{S}}_{k}(m)$

```
$(k, X) \leftarrow S(k)$
return $(k, \mathtt{Tag}_X(m))$
```

Algorithm 5.9: $\mathtt{Vrfy}^{\mathtt{S}}_{k'}(m, t)$

```
$(k', X) \leftarrow S(k')$
return $(k', \mathtt{Vrfy}_X(m, t))$
```

Construction 5.1. *Let $(K, \mathtt{S})$ be a stateful pseudorandom generator, where $\mathtt{S}\colon \mathbb{K} \to \mathbb{K} \times \mathbb{X}$, and let $\Pi = (K', \mathtt{Tag}, \mathtt{Vrfy})$ be a MAC, where $\mathtt{Tag}\colon \mathbb{X} \times \mathbb{M} \to \mathbb{T}$. Then $\Pi^{K,\mathtt{S}}$ is the stateful MAC $(K^2, \mathtt{Tag}^{\mathtt{S}}, \mathtt{Vrfy}^{\mathtt{S}})$, where K^2 is defined by $K^2(x, x) = K(x)$ and the circuits are defined as above.*

Theorem 5.7. *Let $(K, \mathtt{S})$, $\Pi = (K', \mathtt{Tag}, \mathtt{Vrfy})$ and $\Pi^{K,\mathtt{S}} = (K, \mathtt{Tag}^S, \mathtt{Vrfy}^S)$ be as defined above. Let $(K, \mathtt{S})$ be λ-leakage resilient (ε, s, α)-forward secure for q queries. Let $(U_{\mathbb{X}}, \mathtt{Tag}, \mathtt{Vrfy})$ be $(\varepsilon', s', 2)$-secure (as defined in definition 5.1).*

Then $\Pi^{K,\mathtt{S}}$ is λ-leakage $(q(\varepsilon + 2^{\alpha}\varepsilon'), s'')$-forward secure for q queries, where Here, s'' is such that, for all tuples $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $|\mathcal{A}_1| + |\mathcal{A}_2| \leq s''$, $|\mathtt{Forge}^{q,\lambda}_{\mathcal{A},\Pi^{K,\mathtt{S}}}| \leq \min(s, s') - 2\max_{k \in \Omega_K} |k|$.[1]

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be as above. Let us consider $\mathtt{Forge}^{q,\lambda}_{\mathcal{A},\Pi^{K,\mathtt{S}}}(K, K)$ as defined in definition 5.6. In particular, what is the probability that $\mathcal{A}_1$'s i-th query first breaks the scheme, i.e. that the i-th query is a query (m, t) such that $\mathtt{VrfyO}_{k'}(m, t)$ sets $r \leftarrow 1$ for the first time?

Note that $\mathcal{A}_1$ would have gotten the same responses to previous queries if we had replaced $\mathtt{VrfyO}$ by a version that only gives out trivial answers (i.e. evaluates the function $\left[(i_{\mathtt{Vrfy}}, m, t) \in \{q'_{1..3}\colon q' \in Q\}\right]$) for the first $i-1$ queries. Since $\mathcal{A}_1$ could easily simulate this function, and such an adapted version $\mathcal{A}'_1$ would still satisfy $|\mathcal{A}'_1| + |\mathcal{A}_2| \leq s$, having access to (this simplified version of) $\mathtt{VrfyO}$ does not meaningfully help the adversary.

Let $\mathbb{X}$ be as defined by $\mathtt{S}\colon \mathbb{K} \to \mathbb{K} \times \mathbb{X}$. Let s' be s'' minus the size of the part of $\mathcal{A}_1$ that must be executed to find the i-th query; let $\mathcal{A}'_1$ be a circuit executing this same part, replacing the first $i-1$ calls to $\mathtt{VrfyO}$ in the above fashion and handling calls to $\mathtt{TagO}$ by sending a request to the oracle $\mathtt{SO}^{\lambda}_{k}(f)$ as defined for algorithm 4.1. Then $H^{\mathrm{HILL}}_{\varepsilon,s'}(X_i|Q) \geq H^{\mathrm{HILL}}_{\varepsilon,s-|\mathcal{A}'_1|}(X_i|X_1, X_2, \ldots, X_{i-1}, \Lambda_{[1..(i-1)\lambda]}) \geq \log(|\mathbb{X}|) - \alpha$. We apply theorem 5.3 to prove that $\mathcal{A}_1$ has probability at most $\varepsilon + 2^{\alpha}\varepsilon'$ of breaking the scheme in round i.

[1]As a technical detail, we execute **return**(0) as soon as the adversary makes the $i+1$-th query and thereafter remove any gates that cannot change the outcome for any input. If we did not do this, we would have to set s'' ridiculously low to account for an adversary $\mathcal{A}'$ composed entirely of $\mathtt{TagO}$ gates, which would make $\mathtt{Forge}^{q,\lambda}_{\mathcal{A},\Pi^{K,\mathtt{S}}}$ at least $|\mathcal{A}'| \cdot |\mathtt{TagO}|$ gates large.

Let us now consider $\mathcal{A}_2$. We will assume that $\mathcal{A}_2$ makes the i-th query to `VrfyO`, including those queries made by $\mathcal{A}_1$, and that this query is the first to break the scheme. By the same argument as above, $H^{\text{HILL}}_{\varepsilon,s'}(X_i|Q,k) \geq H^{\text{HILL}}_{\varepsilon,s-|\mathcal{A}'_1|-|\mathcal{A}'_2|}(X_i|X_1, X_2, \ldots, X_{i-1}, \Lambda_{[1..(i-1)\lambda]} \geq |\mathbb{X}| - \alpha$ and therefore $\mathcal{A}_2$ has probability at most $\varepsilon + 2^\alpha \varepsilon'$ of breaking the scheme in this round. Here, $\mathcal{A}'_1$ is as above, making as many queries as $\mathcal{A}_1$ did; s' is $s'' - |\mathcal{A}_1|$ minus the size of the part of $\mathcal{A}_2$ that must be evaluated to find the i-th query; and $\mathcal{A}'_2$ is that same part, replacing calls to `VrfyO` as $\mathcal{A}'_1$ does. $\square$

There are several interesting open problems in this space; see section 6.3.

6 Open questions

We give a succinct, expert-level overview of some of the more important open questions related to this thesis. Leakage-resilient cryptography is a very new field, so it is perhaps unsurprising that the number of open questions is rather large, even for such a basic object. Still, even fairly obvious questions remain unanswered.

6.1 Modelling leakage

The model presented in section 3.2 is not the only model being studied, and there is a lively debate both in the theory community and between the theory community and the practitioners on the best way to model side-channel attacks. Pietrzak gives a good overview of the various theoretical models [Pie09b]; we have tried to give an overview of various real-world attacks in section 3.1. This material is based on talks and informal discussions at the workshop “Provable Security against Physical Attacks” (15–19 February, Leiden, the Netherlands) and some of my own thoughts.

Choosing the leakage function. Our model allows the adversary to choose the leakage function f. In some real-world attacks, this can actually be done to a certain extent: for instance, when measuring the electromagnetic fields generated by a device, the probe may be moved to measure the power consumption of specific parts. Modelling this does not hurt (the alternative would be to give *all* such information to the adversary, which is perhaps more than can be achieved in real-world attacks.)

On the other hand, one could argue that the adversary is given the ability to do far too precise measurements. As noted in section 4.2, if we want to design a secure pseudorandom generator our model forces us to use at least two parts of state: otherwise, the adversary can compute (part of) some future key $k_i = \mathtt{S}(\mathtt{S}(\ldots \mathtt{S}(k_0)_1 \ldots)_1)_1$. Certainly, side-channel attacks can be distressingly powerful — but could they really be used to obtain such a complex function of the internal state?

Some attempts have been made at reducing the power of the adversary in this respect (see [Pie09b, section 2.4],) but these models may err in the other direction.

Separate calculations leak separately. We assume that separate calculations do not influence the leakage of other calculations *at all*. This is a very strong assumption; in the real world, any effect would be extremely small, but is would be hard to guard against any interference at all.

Only computation leaks information. There have been a number of misunderstandings related to the "only computation leaks information" assumption. In particular, we do not assume that overwriting memory does not cause the data stored there to leak. As noted in section 3.1, overwriting (CMOS) memory uses power related to the number of bits that are different between the old and the new data. In other words, power analysis would tell the attacker something about the previous contents. This is modelled: we allow the attacker to leak from overwritten data as well.

We do assume that we have some storage from which data "at rest" cannot be extracted. Flash storage, as used in USB sticks, may well have this property. Otherwise, shielded hardware may be required. This is not an entirely unreasonable requirement: leakage-resilient cryptography is most likely to find a use on specialized hardware like smartcards, and we only need a very small amount of secure storage for most algorithms.

Cold-boot attacks. Cold-boot attacks are quite different from the attacks we usually consider: they give a large amount of information, but only once, whereas our model talks about a continuous trickle. Nonetheless, (Dziembowski and) Pietrzak's generators [DP08, Pie09a] both require only that the leakage from different operations is uncorrelated (and not the full strength of the "only computation leaks information" assumption.) The data recovered from cold-boot attacks appears to depend only on very "local" properties of the memory, as one would expect (see [HSH$^+$08, section 3.3 and 5]); thus, we have also proven these algorithms secure against cold-boot attacks, albeit only cold-boot attacks that recover a very small part of the key. So we can model these attacks, but we do not resist them very well.

Leaking λ bits. In almost all cases, there is a very real problem with our model: what does it *mean* to "leak λ bits"? Even basic power analysis attacks will produce a lot of data — very likely, the amount of data read per invocation of the algorithm is much greater than the length of any private keys involved. This data, of course, is not the short but high-signal data that we consider in our theory, but rather a lot of very noisy data about all bits at once (instead of one bit in particular).

Tampering attacks are even harder to shoehorn into this model. We do not consider whether a tampering attack may cause the scheme to malfunction in a way that breaks the security notion. For instance, we do not consider whether an adversary may be able to bypass the MAC we construct simply by forcing the output of the verification procedure to 1. Such attacks may be devastating, e.g. when the security of a computing system hinges on loading only trusted software.

Additionally, even minor errors introduced into the computation may break the scheme, as proven by Boneh *et al.* [BDL97]. On the other hand, not all algorithms are equally vulnerable; therefore, the power of tampering attacks depends on the entire scheme being considered, and not just upon, say, the number of bits that can reasonably be flipped.

These issues make our theorems less useful: the scheme may be secure if the implementation leaks at most λ bits, but if that prerequisite cannot be checked the theorem isn't too useful in the real world.

None of the above, however, should be misconstrued as saying that this model is not the best model we have.

6.2 Pseudorandom generators

Dziembowski and Pietrzak have proven the existence of leakage-resilient pseudorandom generators [DP08]. Pietrzak has presented a somewhat refined algorithm [Pie09a] which has the additional property of only relying on a single primitive which can be implemented using a block cipher.

Block ciphers appear to be very solid. The venerable DES algorithm, designed in the seventies, can still only feasibly be cracked by trying every possible key (unfortunately, there are so few keys that this works well, as demonstrated by the Electronic Freedom Foundation [LG98].) No feasible attacks against AES are currently known and industry is very familiar with this algorithm and how to implement it while remaining secure from side-channel attacks. New attacks on AES are being developed, but since Pietrzak's generator is not bound to any particular block cipher this should not present a real problem — as long as any block cipher is secure, this generator is secure.

Nonetheless, there are still quite a few interesting questions in this area.

Unpredictability instead of indistinguishability. Throughout chapter 4, we consider random variables X_i that cannot be distinguished from a random variable with high (maximal) min-entropy. However, most of chapter 5 — with section 5.2 as the clearest example — only uses the fact that the output of a pseudorandom generator is unpredictable, which is a weaker requirement than being indistinguishable from random. This notion is harder to work with, but it may be possible to obtain better bounds by considering unpredictability instead of pseudoentropy.

An equivalent of Yao's theorem. Yao's theorem tells us that if the next bit of output of a pseudorandom generator is hard to predict given the output up to that point, the output is pseudorandom.[1]

[1]It is common to refer to Yao's own work [Yao82] for this theorem, but, strictly speaking, it is not proven there (theorem 3 probably comes closest). Ballet and Rolland do give a proof in

We know that the next "block" of output of (Dziembowski and) Pietrzak's generators [DP08, Pie09a] is pseudorandom (in particular, hard to predict) given previous output and previous leakage (definition 4.6); we also know that the next "block" has high pseudoentropy given previous output and all leakage including the leakage of this round (definition 4.7.)

It would be nice, although not necessary for the work done in this thesis, to prove an equivalent to Yao's theorem for these generators. However, an adversary, once she has obtained X_1, can request $H(X)$, $(X_i)_{1..\lambda}$, $(X_i)_{\lambda+1..2\lambda}$ and so on, where $H(X)$ is some hash or checksum of X, as leakage. The original statement can be proven with a fairly straightforward hybrid argument; this problem, though, appears to be much more difficult.

In addition to the above questions, consider "Bidirectional communication," below.

6.3 MACs

We have constructed a leakage-resilient MAC in section 5.4, but there are still several interesting open problems. Note that almost all of these can be solved by switching to leakage-resilient signatures as proposed by Faust *et al.* [FKPR10] if a reduction in performance is acceptable.

Create a MAC that doesn't desynchronise easily. We proposed a very general construction that is quite secure; unfortunately, since we used a general leakage-resilient pseudorandom generator, we cannot "rewind" and obtain previous output of this generator. This means that an adversary can easily cause the authenticator (`Tag`)) and the verifier (`Vrfy`) to use different keys: simply make the verifier verify any message.

We could embed a counter in the messages and try to verify any message with a counter between the current value and the current value plus a safety margin. This does not significantly decrease security and makes the system a lot more robust (e.g. one can play with the keyfob without being locked out of one's car; apparently KeeLoq employs a similar "window".) On the other hand, it does not help against malicious intent: an attacker can simply send a couple of messages at the far end of the window and thereby make the keyfob "fall behind."

A better solution would be a tree-based scheme, which would allow us to create a stateless verifier. As a bonus, this would allow us to efficiently verify *any* pair of counter and message, provided that no message with this counter has been accepted yet.

Bidirectional communication. Our construction is only secure if only the authenticator leaks — we do not allow the verifier to leak at all. A verifier that

[BR09]; they appear to have obtained it from Goldreich's textbook [Gol01]. I would recommend obtaining a copy of said textbook.

can tolerate leakage would be useful, as it would allow authenticated bidirectional communication. There is some hope that a tree-based construction as hinted at above might work; however, it still presents plenty of problems. Notably, since $\mathtt{Tag}_{X_i}(m)$ quite possibly discloses a lot of information about X_i, the adversary can potentially take (parts of) X_{i+1}, X_{i+2} etcetera into account when requesting leakage for round i. It is fairly easy to see that, for instance, Pietrzak's generator [Pie09a] is not secure in this scenario: the adversary can calculate $k_{i+2} = \mathtt{S}(k_i, X_i)$, $k_{i+4} = \mathtt{S}(k_{i+2}, X_{i+2})$ and so on, and request part of some future key as leakage.

It would also be interesting to consider bidirectional communication where both devices can leak at the same time. Unfortunately, no pseudorandom generator which can withstand this kind of leakage is currently known.

Going further, it may be useful to consider n devices. "Signing" a per-device key (and, optionally, per-device state) using a regular MAC may make it possible to do this efficiently: essentially, we let each device keep the state related to that device for us, and ensure that the state cannot be tampered with by signing it.

Encryption and authentication. Many communication channels must be immune to both eavesdropping and manipulation — that is, we should be able to combine encryption with a message authentication code. In black-box cryptography we can encrypt the data and then authenticate the encrypted data ("encrypt-then-authenticate"); this is guaranteed to be secure if the encryption and MAC are secure (Katz and Lindell treat this at length [KL08, section 4.9].)

In leakage-resilient cryptography, things aren't quite as clear-cut. The obvious analogue is to calculate $(k, X) \leftarrow \mathtt{S}(k)$ (using a leakage-resilient pseudorandom generator) and $\mathtt{Tag}_{k'}(X \oplus m)$ ((using our MAC,) where m is the message and k and k' are keys. However, this has many problems.

For one, our model allows the adversary to obtain part of (k, k', m) while we are operating on it; therefore, the adversary can obtain part of our message. Worse, the adversary can obtain something that is related to both k and k'. If our algorithms are good, both keys will still have high pseudoentropy; however, from the perspective of the adversary, *the keys are no longer independent* — which is required for Katz and Lindell's proof that encrypt-then-authenticate works.

It appears that leakage-resilient encryption has not been extensively studied; however, this question would need to be answered if we want to have a full suite of leakage-resilient protocols.

Appendices

A Notes on existing generators

(Dziembowski and) Pietrzak asserted that their generators were forward-secure (using a different definition than the one we use, definition 4.7; we will call their definition past-leakage resilient forward security.) Unfortunately, the proof relies on an "obvious" property of HILL-pseudoentropy which is likely not true. We point out the problem in section A.1 and prove a lemma that can provide the missing step in section A.2. Finally, we prove that these generators are leakage resilient forward secure (i.e. secure as defined in definition 4.7) in section A.2.

A.1 Pseudoentropy and leakage

It is a well-known (among cryptographers) fact that $H_\infty(X)$ changes in a sensible way if we give out information about X. Intuitively, if $H_\infty(X) = p$ and we give out λ bits, X "has $p - \lambda$ unknown bits left". However, this does not appear to be the case for HILL-pseudoentropy. Let us first give the result for min-entropy, which inspired us to believe that something similar may hold for pseudoentropy.

Proposition A.1. *Let X and V be random variables and let $T = \Omega_V$. Then*

$$P_{V=v}[H_\infty(X_v) \geq H_\infty(X) - \log|T| - \log\Delta] \geq 1 - \frac{1}{\Delta}$$

for all $\Delta \in \mathbb{R}_{>0}$.

Proof. The desired bound follows from a simple calculation:

$$\begin{aligned}
&P_{V=v}[H_\infty(X_v) < H_\infty(X) - \log|T| - \log\Delta] \\
&\quad = \sum_{v\in T} P[V=v]\left[\max_{x\in\Omega_X} P[X=x|V=v] > \Delta|T|2^{-H_\infty(X)}\right] \\
&\quad \leq \sum_{v\in T} P[V=v]\left[\max_{x\in\Omega_X} \frac{P[X=x]}{P[V=v]} > \Delta|T|2^{-H_\infty(X)}\right] \\
&\quad \leq \sum_{v\in T} P[V=v]\left[\frac{2^{-H_\infty(X)}}{P[V=v]} > \Delta|T|2^{-H_\infty(X)}\right]
\end{aligned}$$

$$= \sum_{v \in T} P[V = v] \left[P[V = v] < \frac{1}{\Delta |T|} \right]$$
$$< \frac{1}{\Delta} \qquad \square$$

It seems as if a proof of an analogue to the above statement should proceed along these lines: define random variables X, Y and V, where X is a random variable on S and Y is (ε, s)-indistinguishable from X given V. Let $f\colon S \to \{0,1\}^\lambda$ be a circuit.

Note $H^{\mathrm{HILL}}_{\varepsilon,s}(X|V) \geq H_\infty(Y)$; we'd hope that $P_{f(X)=z}[H^{\mathrm{HILL}}_{\varepsilon',s'}(X|V, f(X)) \geq H_\infty(Y_z) - \log \Delta | f(X) = z] \geq 1 - \Delta$ for $\varepsilon' \approx 2^\lambda \varepsilon$, $s' \approx s - |f|$ and small Δ. We will introduce some standard notions and then show that this simply isn't true.

Definition A.2. A (ε, s)-*one-way function* is a function $F\colon S \to T$ such that, for all circuits $\mathcal{A}$ of size at most s, $P[F(\mathcal{A}(F(U_S))) = F(U_S)] < \varepsilon$.

Note that one-way functions do not necessarily make it hard to recover *part* of their input: if F is a one-way function, so is $F'\colon S \times S' \to T \times S'\colon F'(x, r) = (F(x), r)$. Nonetheless, a one-way function must make it hard to recover some part of the input. We can formalize this "some part" with the notion of hardcore predicates (bits).

Definition A.3. An (ε, s)-*hardcore predicate* (also called a *hardcore bit*) for a function $F\colon \{0,1\}^n \to \{0,1\}^m$ is a function $h\colon \{0,1\}^n \to \{0,1\}$ such that, for all circuits $\mathcal{A}$ of size at most s, $P[\mathcal{A}(F(U_n)) = h(U_n)] \leq \frac{1}{2} + \varepsilon$.

It is not currently known whether one-way functions or hardcore predicates exist. It is widely believed that this is the case, though. However, if we could prove that one-way functions exist then we immediately obtain $\mathbf{P} \neq \mathbf{NP}$ (as noted in section 1.2, this is believed to be true but has not been proven despite numerous attempts.)

Given a one-way function, it is possible to construct a one-way function with a specific hardcore predicate: if $F\colon \{0,1\}^n \to \{0,1\}^m$ is a one-way function, $h(x, r) = \oplus_{i=1}^n x_i \cdot r_i$ is a hardcore predicate for $F'\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m \times \{0,1\}^n\colon F'(x, r) = (F(x), r)$. This was proven by Goldreich and Levin, and is therefore called the Goldreich-Levin theorem [GL89]; Katz and Lindell also treat it [KL08, section 6.3].

Remark A.4. *Let $F\colon \{0,1\}^n \to \{0,1\}^m$ be a one-way function and let $h\colon \{0,1\}^n \to \{0,1\}$ be a hardcore predicate of this function. Pick any $\beta \in [0,1]$, and define*

$$\begin{aligned}
X &= (U_1, U_p, h(U_n)) \\
Y_0 &= (U_1, U_p, U_1') \\
Y_1 &= (h(U_n), U_p, U_1') \\
Y_\beta &= (1 - \beta) Y_0 + \beta Y_1 \\
V &= F(U_n)
\end{aligned}$$

Since h is a hardcore predicate of F, $h(U_n)$ is indistinguishable from U_1 given $V = F(U_n)$.

However, define $f\colon \{0,1\}^{p+2} \to \{0,1\}\colon x \mapsto x_{p+2}$. X and Y are distinguishable given V and *$f(X) = h(U_n)$: consider a circuit D that outputs 1 exactly if the first bit equals $h(U_n)$. We obtain*

$$\begin{aligned}\varepsilon' &\geq |P[D(Y^\beta, V, f(X)) = 1] - P[D(X, V, f(X)) = 1]|\\ &= |P[Y_1^\beta = b(Q,R)] - P[X_1 = b(Q,R)]|\\ &= |(1-\beta)\cdot P[U_1 = z] + \beta - P[U_1 = z]|\\ &= \beta \cdot P[U_1 \neq z]\\ &= \frac{\beta}{2}\end{aligned}$$

Unless β is very small, this means that ε' is much larger than $2^\lambda\varepsilon$.

Note that our definition of Y_β seems pretty innocuous: it has high min-entropy (Y^0 "seems optimal"). Of course, f is utterly trivial. Furthermore, Y_β is *efficiently samplable*, i.e. we can efficiently create a random variable with the same distribution as Y^β using only uniform randomness (for particular, arbitrarily small values of β.) Consider $(1 - [U_q = 0^q])(U_1, U_p, U_1') + [U_q = 0^q]\,(h(U_n), U_p, U_1')$; this has the same distribution as Y_{2^q}. It does not seem possible to restrict the problem further and still hope to obtain a relevant result.

Hsiao *et al.* give a different, and arguably better, concept of conditional min-entropy [HLR07]. They then base their definition of HILL-pseudoentropy on this concept just as we have based our definition of HILL-pseudoentropy on Dziembowski and Pietrzak's concept of computational entropy [DP08]. Unfortunately, using their definition would not help us: remark A.4 suggests that the problem is in maintaining indistinguishability.

A.2 Past-leakage resilient forward security

Despite the above problems, (Dziembowski and) Pietrzak's generators *are* past-leakage resilient forward secure. The following lemma can be used to provide the missing step from the next-to-last equation to the last equation of Dziembowski and Pietrzak's article [DP08]; Pietrzak's generator is not explicitly proven secure, but Pietrzak asserts past-leakage resilient forward security based on essentially the same argument as in the earlier paper, and the same argument applies.

Lemma A.5. *Let X, X', Y, Y' and V be random variables such that X' is independent from Y and Y'. Suppose (X, Y, V) is (ε, s)-indistinguishable from (X', Y', V). Then (X, Y, V) is $(2\varepsilon, s - \max_{x\in\Omega_X}|x|)$-indistinguishable from (X', Y, V).*

Proof. Suppose not, i.e. let $\mathcal{A}$ be a circuit of size at most $s - \max_{x\in\Omega_X}|x|$ such that $|P[\mathcal{A}(X, Y, V) = 1] - P[\mathcal{A}(X', Y, V) = 1]| \geq 2\varepsilon$.

Suppose that $|P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y',V)=1]| \geq \varepsilon$. Since X' is independent from Y and Y', there is an $x^* \in \Omega_X$ such that

$$\begin{aligned}
&|P[\mathcal{A}(x^*,Y,V)=1]-P[\mathcal{A}(x^*,Y',V)=1]| \\
&\quad \geq |P[\mathcal{A}(X',Y,V)=1|X'=x^*]-P[\mathcal{A}(X',Y',V)=1|X'=x^*]| \\
&\quad \geq \varepsilon
\end{aligned}$$

Consider the circuit $\mathcal{A}'(X,Y,V)=\mathcal{A}(x^*,Y,V)$. It would distinguishes between (X,Y,V) and (X',Y',V) with probability at least ε and is of size $|\mathcal{A}'| = |\mathcal{A}| + |x^*| \leq s$. This contradicts our assumption, and therefore $|P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y',V)=1]| < \varepsilon$. But then

$$\begin{aligned}
&|P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y',V)=1]| \\
&\quad = |P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y,V)=1]+ \\
&\qquad P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y',V)=1]| \\
&\quad \geq |P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y,V)=1]|+ \\
&\qquad |P[\mathcal{A}(X',Y,V)=1]-P[\mathcal{A}(X',Y',V)=1]| \\
&\quad > 2\varepsilon - \varepsilon \\
&\quad = \varepsilon
\end{aligned}$$

Therefore, $\mathcal{A}$ would distinguish between (X,Y,V) and (X',Y',V) wth probabilty at least ε. This again contradicts our assumption, so no such $\mathcal{A}$ — no $\mathcal{A}$ that distinguishes between (X,Y,V) and (X',Y,V) — exists. □

A.3 LEAKAGE-RESILIENT FORWARD SECURITY

We prove that a slightly changed version of Pietrzak's generator [Pie09a] is forward secure as defined in definition 4.7. The same argument can be applied to Dziembowski and Pietrzak's earlier generator [DP08]. Be warned that the proof requires a good understanding of Pietrzak's work, most importantly [Pie09a, lemma 7]. If one is willing to assume that generators satisfying definition 4.7 exist, this section may be skipped.

Pietrzak's generator is based on weak pseudorandom functions, so let us introduce them.

Definition A.6 ([Pie09a, section 1.2]). Let $F\colon \{0,1\}^\kappa \times \{0,1\}^n \times \{0,1\}^m$ be a function; let $K = U_\kappa$; draw $R\colon \{0,1\}^n \to \{0,1\}^m$ uniformly at random from the functions from $\{0,1\}^n$ to $\{0,1\}^m$. Then F is a (ε,s,q)*-secure weak pseudorandom function* if $(F_K(U_n^{(1)}), F_K(U_n^{(2)}), \ldots, F_K(U_n^{(q)}))$ is (ε,s)-indistinguishable from $(R(U_n^{(1)}), R(U_n^{(2)}), \ldots, R(U_n^{(q)}))$.

Following Dziembowski and Pietrzak's work, we split the output X_i into two parts: X_i^{out} remains output and X_i^{nxt} is used in the further calculation.

Definition A.7. Let $F\colon \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^\kappa \times \{0,1\}^n \times \{0,1\}^m$ be a weak pseudorandom function. Consider the pseudorandom generator

$((0, U_\kappa, U_\kappa, U_n), \mathtt{S})$. If i is even, we define $\mathtt{S}(i, K, K', X^{\mathrm{nxt}}) = (i+1, F(K, X^{\mathrm{nxt}})_1, K', F(K, X^{\mathrm{nxt}})_2, F(K, X^{\mathrm{nxt}})_3)$; if i is odd, we define $\mathtt{S}(i, K, K', X^{\mathrm{nxt}}) = (i+1, K, F(K', X^{\mathrm{nxt}})_1, F(K, X^{\mathrm{nxt}})_2, F(K, X^{\mathrm{nxt}})_3)$.

We can now prove that this variant is leakage-resilient forward secure. The proof is not too complex, but relies on many details of that article.

Theorem A.8. *Let (U_κ, F') be the algorithm defined above. Suppose that F is a $(\varepsilon, s, \frac{n}{\varepsilon})$-secure weak pseudorandom function. Let $\varepsilon \geq n \cdot 2^{\frac{-n}{3}}$, $n \geq 20$, $\lambda = \frac{\log(\varepsilon^{-1})}{6}$. Then (U_κ, F') is λ-leakage resilient $(8(q+1)\varepsilon^{\frac{1}{12}}, \frac{s\varepsilon^2}{2^{\lambda+2}(n+\kappa)^3} - 2\kappa + n, q, 2\lambda)$-forward secure.*

Proof. Let all variables be as in the proof of [Pie09a, lemma 7]. Define K'_i analogously to K'_{i-1} by demanding that K'_i is indistinguishable from K_i given $\mathtt{view}_i^-$. Define $X_i'^{\mathrm{nxt}} = (X'_i)_{1..n}$ and $X_i'^{\mathrm{out}} = (X_i)_{n+1..n+m}$. Note that these can all be chosen to be independent; let us do so. Now, K'_i and K'_{i-1} have min-entropy $\kappa - 2\lambda$, $X_i'^{\mathrm{nxt}}$ has min-entropy $n - 2\lambda$ and $X_i'^{\mathrm{out}}$ has min-entropy $m - 2\lambda$ (there is a typographical error in the lemma, which states $H_\infty(K'_i) \geq \kappa - \lambda$ etcetera, omitting the "2.") As noted in the proof of lemma 7, $H^{\mathrm{HILL}}_{\varepsilon', s'}(F(K'_{i-1}, X'^{\mathrm{nxt}}_{i-1}) | \mathtt{view}_i^-) \geq \kappa + n + m - 2\lambda$.

There are K''_{i+1}, $X_i''^{\mathrm{nxt}}$ and $X_i''^{\mathrm{out}}$ with min-entropy $\kappa - 2\lambda$, $n - 2\lambda$ respectively $m - 2\lambda$ that are (ε', s')-indistinguishable from $F(K'_{i-1}, X'_{i-1}) = (K'_{i+1}, X_i'^{\mathrm{nxt}}, X_i'^{\mathrm{out}})$ (see [Pie09a, lemma 6], near the end of the proof; we use ε' and s' instead of the exact value to simplify the expressions we obtain.) Furthermore, we can choose these to be independent from each other and K'_i. Therefore, $H^{\mathrm{HILL}}_{\varepsilon', s'-2\kappa+n}(X_i''^{\mathrm{out}} | \mathtt{view}_i^-, K'_i, K''_{i+1}, X_i''^{\mathrm{nxt}}) \geq m - 2\lambda$. Returning to the actual variables, $H^{\mathrm{HILL}}_{\varepsilon_i+2\varepsilon', s_i-2\kappa+n}(X_i^{\mathrm{out}} | \mathtt{view}_i^-, K_i, K_{i+1}, X_i^{\mathrm{nxt}}) \geq m - 2\lambda$, where ε_i and s_i are as in the article.

We sum the error in the same fashion as in [Pie09a], adding the error ε' and subtracting the circuit size $2\kappa + n$ introduced above. □

B An extractor-based approach

We initially tried to construct an authentication protocol based on a randomness extractor. We were not able to prove it secure; this appendix presents the problem that caused us to look for other approaches, culminating in construction 5.1 on page 29. randomness extractor effectively turns a small random seed and a distribution with sufficient entropy into a uniform distribution.

Definition B.1. A function $e\colon \mathbb{K} \times S \to T$ is a (p, ε)*-randomness extractor* if, for all random variables K over $\mathbb{K}$ with min-entropy at least p, $e_K(X)$ is *(statistically)* ε*-close* to U_T, i.e. if $\frac{1}{2} \sum_{t \in T} |P[e_K(X) = t] - P[U_T = t]| \leq \varepsilon$.

Note that the output of an extractor is "random" in a statistical sense, not just in a computational sense. We hoped that this would make it easier to compose the extractor and the pseudorandom generator into an authentication protocol.

On the other hand, note that the output of the extractor is only guaranteed to be random for random X: it's entirely possible that $e_k(0^n) = 0^m$ for all k, and likewise $e_k(0^n)$ and $e_k(0^{n-1}||1)$ may be very similar for all k.

Our proof of theorem 5.7 requires that the adversary almost never gets a non-trivial positive response from `VrfyO`. Unfortunately, since the extractor may produce similar output for similar input, we cannot guarantee this: the adversary may be able to generate a tag on a message related to but not equal to the message sent to `TagO`. Even if this could be worked around by changing the security definition, such a leakage-resilient MAC would not be secure in the black-box sense definition 2.5.

It may be possible to make this work by having the verifier sign challenges (using public-key cryptography) and having the authenticator only respond to correctly signed challenges. However, this would result in a rather unwieldy protocol based on several unrelated algorithms; the advantage over using the leakage-resilient public-key system introduced by Faust *et al.* [FKPR10] would be much smaller. We did not pursue this further once these problems became apparent.

Bibliography

[Adl78] Leonard M. Adleman.
Two theorems on random polynomial time.
In *FOCS*, pages 75–83. IEEE, 1978.

[BB05] David Brumley and Dan Boneh.
Remote timing attacks are practical.
Computer Networks, 48(5):701–716, 2005.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton.
On the importance of checking cryptographic protocols for faults (extended abstract).
In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[Boi06] Adam Boileau.
Hit by a bus: Physical access attacks with Firewire.
`http://www.storm.net.nz/static/files/ab_firewire_rux2k6-final.pdf`, 2006.

[BR09] Stéphane Ballet and Robert Rolland.
A note on a Yao's theorem about pseudorandom generators.
Cryptology ePrint Archive, Report 2009/548, 2009.
`http://eprint.iacr.org/`.

[BSW03] Boaz Barak, Ronen Shaltiel, and Avi Wigderson.
Computational analogues of entropy.
In Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *RANDOM-APPROX*, volume 2764 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2003.

[BY03] Mihir Bellare and Bennet S. Yee.
Forward-security in private-key cryptography.
In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2003.

[CWR09] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi.
Opportunities and limits of remote timing attacks.
ACM Transactions on Information and System Security, 12(3), 2009.

[DH76] Whitfield Diffie and Martin E. Hellman.
New directions in cryptography.

IEEE Transactions on Information Theory, 22:644–654, 1976.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak.
Leakage-resilient cryptography.
In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[EKM$^+$08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani.
On the power of power analysis in the real world: A complete break of the KeeLoq code hopping scheme.
In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum.
Leakage-resilient signatures.
In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2010.

[Gir10] Christophe Giraud.
Practical difficulties of physical attacks.
Presentation at "Provable Security against Physical Attacks," Leiden, The Netherlands, February 15-19, 2010.
Slides available at `http://www.lorentzcenter.l/lc/web/2010/383/presentations/Giraud.ppt`.

[GL89] Oded Goldreich and Leonid A. Levin.
A hard-core predicate for all one-way functions.
In *STOC*, pages 25–32. ACM, 1989.

[Gol01] Oded Goldreich.
Foundations of Cryptography, volume 1.
Cambridge University Press, Cambridge, United Kingdom, 2001.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby.
A pseudorandom generator from any one-way function.
SIAM Journal of Computing, 28(4):1364–1396, 1999.

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin.
Conditional computational entropy, or toward separating pseudoentropy from compressibility.
In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 2007.

[HSH$^+$08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten.
Lest we remember: Cold boot attacks on encryption keys.
In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 45–60. USENIX Association, 2008.

[IKD^{+}08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel.
A practical attack on KeeLoq.
In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[KL08] Jonathan Katz and Yehuda Lindell.
Introduction to modern cryptography: principles and protocols.
Cryptography and network security. Chapman & Hall/CRC, Boca Raton, Florida, USA, 2008.

[Knu92] Donald E. Knuth.
Two notes on notation.
The American Mathematical Monthly, 99(5):403–422, 1992.

[Law09] Nate Lawson.
Timing attack in Google Keyczar library.
http://rdist.root.org/2009/05/28/timing-attack-in-google-keyczar-library/, 2009.

[LG98] Mike Loukides and John Gilmore, editors.
Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design.
O'Reilly & Associates, Inc., Sebastopol, California, USA, 1998.

[MR04] Silvio Micali and Leonid Reyzin.
Physically observable cryptography (extended abstract).
In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.

[Pap94] Christos Papadimitriou.
Computational Complexity.
Addison-Wesley, Reading, Massachusetts, USA, 1994.

[Per05] Colin Percival.
Cache missing for fun and profit.
http://www.daemonology.net/hyperthreading-considered-harmful/, 2005.

[Pie09a] Krzysztof Pietrzak.
A leakage-resilient mode of operation.
In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.

[Pie09b] Krzysztof Pietrzak.
Provable security for physical cryptography.
`http://homepages.cwi.nl/~pietrzak/publications.html`, 2009.
Accompanies a survey talk given at the Western European Workshop on Research in Cryptology 2009, WEWORC 2009, July 7-9, 2009, Graz, Austria.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
Communications of the ACM, 21(2):120–126, 1978.

[RTTV08] Omer Reingold, Luca Trevisan, Madhur Tulsiani, and Salil P. Vadhan.
Dense subsets of pseudorandom sets.
Electronic Colloquium on Computational Complexity (ECCC), 15(045), 2008.

[Yao82] Andrew Chi-Chih Yao.
Theory and application of trapdoor functions (extended abstract).
In *FOCS*, pages 80–91. IEEE Computer Society, 1982.

Index