Automation and Modularity of Cryptographic Proofs in the Computational Model

PhD Thesis

Juan Manuel Crespo
Automation and Modularity of Cryptographic Proofs in the Computational Model

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software and Systems

Author: Juan Manuel Crespo
Advisor: Gilles Barthe
Disclaimer

This dissertation builds on several published works that I have co-authored.

Conference articles:

- Fully automated analysis of padding-based encryption in the computational model.
  With Gilles Barthe, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin.

- Mind the Gap: Modular Machine-Checked Proofs of One-Round Key Exchange Protocols
  With Gilles Barthe, Yassine Lakhnech, and Benedikt Schmidt.

I contributed in the elaboration of all of them, as well as in the development of the supporting machine-checked proofs.

EasyCrypt developments accompanying this dissertation are available upon request.
Conference articles co-authored during the term of my Ph.D. grant, but
not included in the dissertation:

- *A Machine-Checked Framework for Relational Separation Logic*
  With César Kunz.
  In *9th International Conference on Software Engineering and Formal Methods - SEFM 2011*

- *Relational Verification Using Product Programs*
  With Gilles Barthe, and César Kunz.
  In *17TH International Symposium on Formal Methods - FM 2011*

- *Secure Multi-Execution through Static Program Transformation*
  With Gilles Barthe, Dominique Devriese, Frank Piessens, and Exequiel Rivas
  In *The 2012 IFIP Joint International Conference on Formal Techniques for Distributed Systems (32rd FORTE / 14th FMOODS) - FMOODS/FORTE 2012*

- *From relational verification to SIMD loop synthesis*
  With Gilles Barthe, Sumit Gulwani, César Kunz, and Mark Marron
  In *18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming - PPoPP 2013*

- *Computer-Aided Cryptographic Proofs.*
  With Gilles Barthe, Benjamin Grégoire, César Kunz, and Santiago Zanella Béguelin.
  In *Third Conference on Interactive Theorem Proving - ITP 2012.*

- *Beyond 2-Safety: Asymmetric Product Programs for Relational Program Verification*
  With Gilles Barthe, and César Kunz
  In *Symposium on Logical Foundations of Computer Science - LFCS 2013*
# Contents

## 1 Introduction

1.1 Security Analysis of Cryptographic Systems ........................................ 3  
1.2 A primer on code-based cryptography proofs ..................................... 5  
1.3 Challenges ......................................................................................... 8  
1.4 Contributions .................................................................................... 9  

## 2 Automated analysis of padding-based encryption  

2.1 Cryptographic Background ............................................................... 13  
\hspace{0.5cm} 2.1.1 Padding-Based Public-Key Encryption .................................. 13  
\hspace{0.5cm} 2.1.2 Security of Padding-Based Encryption Schemes ....................... 14  
2.2 Symbolic Tools .................................................................................. 15  
\hspace{0.5cm} 2.2.1 Syntax and Semantics of Expressions .................................... 15  
\hspace{0.5cm} 2.2.2 Equality and Deducibility ....................................................... 16  
2.3 Chosen-Plaintext Security .................................................................. 17  
\hspace{0.5cm} 2.3.1 Judgments .............................................................................. 18  
\hspace{0.5cm} 2.3.2 Rules ...................................................................................... 18  
\hspace{0.5cm} 2.3.3 Bounds .................................................................................... 21  
\hspace{0.5cm} 2.3.4 Soundness .............................................................................. 22  
\hspace{0.5cm} 2.3.5 Examples .............................................................................. 22  
2.4 Chosen-Ciphertext Security ............................................................... 25  
\hspace{0.5cm} 2.4.1 Judgments .............................................................................. 25  
\hspace{0.5cm} 2.4.2 Rules ...................................................................................... 26  
\hspace{0.5cm} 2.4.3 Bounds .................................................................................... 28  
\hspace{0.5cm} 2.4.4 Soundness .............................................................................. 29  
\hspace{0.5cm} 2.4.5 Example .............................................................................. 30  
2.5 Attacks .............................................................................................. 32  
\hspace{0.5cm} 2.5.1 Inequality and Static Distinguishability .................................. 32  
\hspace{0.5cm} 2.5.2 Universal Attacks ................................................................. 34  
\hspace{0.5cm} 2.5.3 Existential Attacks ............................................................... 34  
2.6 Experimental Validation .................................................................... 35  
\hspace{0.5cm} 2.6.1 Security Analysis ................................................................. 35  
\hspace{0.5cm} 2.6.2 Generation of Encryption Schemes ....................................... 37  
\hspace{0.5cm} 2.6.3 Experiments .......................................................................... 37
## Contents

2.7 Proofs ................................................................. 40
   2.7.1 Soundness of CPA logic ........................................ 40
   2.7.2 Soundness of CCA logic ........................................ 45

3 Modular Proofs of Key Exchange Protocols 51
   3.1 Background .......................................................... 55
      3.1.1 Notation ....................................................... 55
      3.1.2 One-Round Authenticated Key Exchange Protocols .......... 56
      3.1.3 EasyCrypt .................................................... 57
      3.1.4 Trapdoor Test, Twin DH, and (S)CDH ....................... 58
   3.2 Model and Generic Proof ........................................... 60
      3.2.1 eCK$_{kr}$ Security and eCK$_{nkr}$ Security ................. 60
      3.2.2 Generic Proof ............................................... 64
   3.3 Case Studies ........................................................ 69
      3.3.1 Proofs for Naxos and Naxos$^+$ .............................. 69
      3.3.2 Proofs for Nets .............................................. 73
   3.4 Protocols Without Naxos Trick .................................... 73
      3.4.1 Model and Generic Proof ..................................... 74
      3.4.2 eCK$_{kr}$-security of mHMQV under the Gap-CDH assumption . . . 74

4 Related Work ......................................................... 77

5 Conclusion ........................................................... 81

Bibliography .......................................................... 94
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>IND-CPA experiment for BR encryption scheme and OW assumption.</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>Sequence of games for IND-CPA security proof of BR.</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Equational theory for algebraic expressions</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Generic CPA game for $c^* : \phi$ on the right and instantiated CPA game for initial OAEP judgment $f(G(r) \oplus (m</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Proof rules of the CPA logic (we use $dist(\vec{e})$ to denote that $\vec{e}$ consists of distinct expressions and $\vec{e}$ to denote the concatenation of its elements when used as an expression)</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Computation of concrete security bounds from logical derivations</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Derivations for CPA security of OAEP under one-wayness and partial-domain one-wayness</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>A derivation for CPA security of PSS-E</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>Proof rules for CCA judgments and proof rules for tests</td>
<td>27</td>
</tr>
<tr>
<td>2.8</td>
<td>Proof rules of the extended CPA logic</td>
<td>28</td>
</tr>
<tr>
<td>2.9</td>
<td>Extended computation of concrete security bounds for CCA derivations</td>
<td>29</td>
</tr>
<tr>
<td>2.10</td>
<td>A derivation for CCA security of PSS-E</td>
<td>30</td>
</tr>
<tr>
<td>2.11</td>
<td>Proof rules for apartness $#$</td>
<td>33</td>
</tr>
<tr>
<td>2.12</td>
<td>Security analysis workflow</td>
<td>36</td>
</tr>
<tr>
<td>3.1</td>
<td>Generic two pass protocol. Protocols using the Naxos trick use boxed messages.</td>
<td>57</td>
</tr>
<tr>
<td>3.2</td>
<td>Twin DDH games $G_{2DDH}$ and $G$.</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>Games $eCK_{kr,II}(A)$ and $eCK_{nkr,II}(A)$ for $A = (A_1, A_2)$ and protocol II.</td>
<td>63</td>
</tr>
<tr>
<td>3.4</td>
<td>Freshness condition for a trace $evs$ and a test session $t = (r, X, \hat{A}, \hat{B}, Y)$.</td>
<td>64</td>
</tr>
<tr>
<td>3.5</td>
<td>Games defining $CSK_{kr,II}(B)$ and $CSK_{nkr,II}(B)$ for $B = (B_1, B_2, B_3)$ with alternative $eqS$-oracle definitions for $kr$ and $nkr$.</td>
<td>66</td>
</tr>
<tr>
<td>3.6</td>
<td>Intermediate game $GI$ used in reductions ($eCK_{nt}$ to $CSK_{nt}$ and $eCK$ to $CSK$).</td>
<td>68</td>
</tr>
<tr>
<td>3.7</td>
<td>Games defining $CSK_{kr}$.</td>
<td>74</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Evaluation of the tool on generated encryption schemes, CPA security . . . . 38
2.2 Evaluation of the tool on generated encryption schemes, CCA security . . 39
2.3 Results on variants of OAEP . . . . . . . . . . . . . . . . . . . . . . . . . . 40

3.1 Obtained proofs for Key Exchange Protocols (*see explanation, nt = non-tight). 55
3.2 Generic Protocol Model with Naxos\textsuperscript{corr} instantiation and transformation. . . 61
Over the last few decades, our civilization has undergone one of the major technological advances of its history: the widespread adoption of computer systems in every aspect of our daily life. This process started with the personal computer (PC), that succeeded in making a technology that had been up to that point reserved for university and government use, readily available to everyone, at an accessible cost. The Internet has accelerated the adoption of the PC as a universal appliance found in every household. This has in turn led to an increasing number of online services such as shopping and home-banking. In addition, Social networks have changed the way in which we communicate, interact and share information with people around the world in a dramatic way. The use of smartphones, which have evolved to become essentially pocket computers, has further increased the use of the Internet and enabled location-aware services and advertisement. We can foresee that wearable devices and the so called “Internet of things” is going to increase even further our dependence on the Internet.

Whereas all these technological advances are exciting and have provided plenty of opportunities for companies and individuals, they have also raised very valid security concerns. Indeed, more and more of our information is stored online: banking information, pictures and videos, location information, communication with relatives and friends. But even more crucial, companies have built huge networks that are connected to the Internet in one way or another and within these networks they share confidential information of every sort: business strategies, product information, patent application information, internal communications, customer information, etc. A natural question that arises is how to protect this information from unauthorized use.

Information Security is the practice of building and implementing counter-measures for defending information from threats. The Reference Model of Information Assurance and Security [44] defines a set of eight security goals:

- Confidentiality: a system should ensure that only authorized users access information.
Chapter 1. Introduction

– Integrity: a system should ensure completeness, accuracy and absence of unauthorized modifications in all its components.

– Availability: a system should ensure that all system’s components are available and operational when they are required by authorized users.

– Accountability: a system should be able to hold users responsible for their actions.

– Non-repudiation: a system should be able to prove occurrence/non-occurrence of an event or participation/non-participation of a user in an event.

– Auditability: a system should be able to conduct persistent, monitoring of all actions performed by humans or machines within the system.

– Authenticity & Trustworthiness: a system should be able to verify identity and establish trust in a third party and in the information it provides.

– Privacy: A system should obey privacy legislation and it should enable individuals to control, where feasible, their personal information.

The tools used to ensure these goals, can be of different nature: legal, technical, organizational, etc. One of the most fundamental technical tools is cryptography. At its core, cryptography provides means to render a piece of information unusable to unauthorized recipients through a process called “encryption”. The information can in turn be recovered by an authorized user (in possession of a key) through “decryption”. This basic functionality is used to design complex security protocols that play an important role in several of the security goals described above. For example, Digital Signature schemes are used to demonstrate the authenticity of a message or document and play a crucial role in achieving authentication, integrity and non-repudiation.

The design and implementation of secure cryptographic primitives and protocols is an extremely difficult task. There is a vast number of examples where poorly designed protocols or flawed implementations have resulted in insecure systems.

A paramount example of a flawed cryptographic protocol is the Needham-Schroeder authentication protocol, introduced in 1978 [85]. The objective of this protocol is to provide means whereby a pair of agents can be assured of each other’s identity in the context of a public network. This protocol was believed to be secure until 1995, when Gavin Lowe discovered a man-in-the-middle attack [80]. Using this attack, an intruder can impersonate an agent $A$ during a run of the protocol, to trick another agent $B$ into thinking that he really is talking to $A$. Lowe also proposed a fix that renders the protocol secure.

Another well-known example of poorly designed cryptographic system, perhaps of greater practical impact, is Wired Equivalent Privacy (WEP). WEP was introduced in 1999 and its purpose was to provide data confidentiality to wireless networks on par with traditional wired networks. The protocol uses stream cipher $RC4$ for confidentiality and
1.1. Security Analysis of Cryptographic Systems

the CRC-32 checksum for integrity. During the beginning of the last decade, WEP became the de-facto standard for wireless security and it was widely deployed, even after it was deprecated by the IEEE in 2004. A number of attacks on the protocol were published [98, 86, 34] and implemented in tools like Aircrack-ng. Due to its many weaknesses, WEP security is seldom used nowadays.

On the other hand, implementation flaws in cryptographic systems provide a huge attack surface for malicious parties. For example, in April of 2014 a bug in the implementation of the OpenSSL cryptography library was disclosed. This bug, coined “Heartbleed” \(^1\), can be exploited regardless of whether the vulnerable implementation is present in the client or in the server. Its exploitation could provide an attacker with unencrypted data exchanged between the parties, authentication information such as session cookies or passwords, and even private keys of the compromised parties. In this last scenario, the attacker can not only decrypt future communications but also past ones in case he has been passively eavesdropping and storing traffic, unless perfect forward secrecy was used. This last fact also highlights that the study of the properties of such widespread cryptographic protocols is of paramount practical importance.

An additional source of vulnerabilities has been the limitations on the implementations of cryptographic systems imposed by U.S. cryptography export regulations. Recently, an attack on TLS that exploits precisely such a weakness was discovered. This attack, coined FREAK \(^2\), targets a class of deliberately weak export cipher suites that is typically disabled by default in most TLS implementations. However, some TLS implementations allow to use an export cipher-suite even if a non-export cipher-suite was negotiated. Export RSA moduli must be less than 512 bits long and can therefore be factored in a short period of time with reasonable computational resources.

In this context, in which every week a new attack is discovered or a seemingly secure system is exploited, there is little doubt that critical components, such as cryptographic systems, need to be formally analyzed and proved secure, both at the design and at the implementation level.

1.1 Security Analysis of Cryptographic Systems

Over the years, two different models for analyzing the security of (the design of) cryptographic constructions have been developed and coexist until today, each with its own advantages and disadvantages: the computational model and the symbolic model.

The symbolic model, which originates from the seminal work of [56], views cryptographic primitives as function symbols in a term algebra of expressions. The properties of primitives and the capabilities of adversaries are modeled using equational theories. While symbolic methods are more amenable to automation, they assume “perfect cryptography” hence providing weak guarantees, in particular, they usually do not capture all “realistic”

\(^1\)http://heartbleed.com/
\(^2\)https://www.smacktls.com/#freak
attacks.

On the other hand, the computational model finds its roots in the work of [62] and views cryptographic primitives as functions on bitstrings. Adversaries are modeled as probabilistic algorithms and security is defined in terms of their success probability and computational resources. Security proofs exhibit reductions, showing how a successful attack can be converted into an efficient algorithm that breaks the underlying assumption. This stronger notion of security often comes at the price of automation.

In order to tame the complexity of proofs in the computational model, the game playing technique [95, 31] was introduced. The crux of this technique is to use of pseudo-code to describe the probabilistic experiments representing the security properties and the underlying assumptions, and structuring the proofs as sequence of such programs. Each transition in the sequence is justified by an informal argument stating that the distributions that the programs—which in cryptography jargon are referred to as games—represent is the same.

Shai Halevi [63] proposed to take this approach one step further and suggested it should be possible to actually formalize the semantics of the programming language and use methods from programming language theory to build a tool to automatically check the transitions at each step of the proof.

CertiCrypt [17] is a formalization of a probabilistic imperative language—akin to the one used in pen-and-paper, game-based cryptographic proofs in the computational model—in the Coq proof assistant. Essentially, it uses methods from programming language theory and program verification to define a formal language that can be used describe games, and to relate the probabilities of events in these games. This development has proved to be useful in setting solid theoretical foundations for computational proofs of cryptography: security properties and cryptographic assumptions are stated explicitly in a uniform language and proofs provide “end to end” guarantees. Several well-known results were formalized in CertiCrypt, e.g. semantic security of OAEP in the random oracle model [16], existential unforgeability of FDH signatures [99], sigma protocols [18], hashing into elliptic curves [14] and indentity based encryption [20].

Although the formalism adheres to the common practice in cryptographic proofs in many aspects, the proof burden for a machine-checked proof within Coq makes it extremely hard to apply this approach to large scale cryptographic systems. Moreover, a high degree of Coq expertise is required to use the tool, making it accessible to a rather limited community.

EasyCrypt [13] was introduced later and partially addressed this issue. The system is not built within a theorem prover, but rather by implemented as a stand-alone tool. Essentially, the language and the logic are similar to the one used in CertiCrypt, but proof obligations are collected and sent to SMT solvers, therefore leveraging on extensive work on automated theorem proving. The tool was effectively used to prove the Cramer-Shoup public key encryption scheme secure against adaptive chosen-ciphertext attacks under standard assumptions, a new redundancy free encryption scheme coined ZAEP [21] and the Merkel-Damgård construction [6].
1.2. A primer on code-based cryptography proofs

For the sake of concreteness, and to better illustrate the challenges addressed by this thesis, we introduce a simple game-based proof of security for a standard encryption scheme. A public key encryption scheme comprises a key generation $\mathcal{KG}$, an encryption $\mathcal{E}$ and a decryption algorithm $\mathcal{D}$. From these, only decryption is deterministic. The idea is that for any valid key pair $(pk, sk)$, the secret key $sk$ can be used to recover a message from a ciphertext obtained by encrypting it using the public key $pk$, i.e. $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$. A prominent example is the encryption scheme introduced by Bellare and Rogaway in [29] and the encryption function is defined as $\mathcal{E}_{BR}^{pk}(m) = f_{pk}(r) \parallel m \oplus H(r)$, where $\parallel$ is bitstring concatenation, $\oplus$ is exclusive or, $r$ denotes a randomly sampled bitstream, $H$ denotes a hash function and $f$ denotes a one way trapdoor permutation, i.e. a function that is cheap to compute, but hard to invert without knowing the secret key.

While the notion of correctness of an encryption scheme given above is simple and intuitive, when one starts thinking what it should mean for an encryption scheme to be secure the situation changes. There are two standard notions of security for encryption schemes: IND-CPA and IND-CCA$^3$. The former captures the idea that a passive adversary should only be able to distinguish the result of the encryption under an arbitrary secret key of two messages of his choice—even when he has access to the matching public key—with negligible probability. The latter captures an active adversary that has also access a decryption oracle and can perform arbitrary queries—except for the challenge ciphertext. In both cases, the adversary also has access to the all the hash functions used. More

---

$^3$We use IND-CCA for the adaptive version usually referred to as IND-CCA2.

<table>
<thead>
<tr>
<th>Game IND-CPA($\mathcal{A}_1, \mathcal{A}_2$):</th>
<th>Game OW($\mathcal{I}$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(pk, sk) \leftarrow \mathcal{KG}();$</td>
<td>$(pk, sk) \leftarrow \mathcal{KG}();$</td>
</tr>
<tr>
<td>$(m_0, m_1) \leftarrow \mathcal{A}_1^H(pk);$</td>
<td>$r \leftarrow {0, 1}^n;$</td>
</tr>
<tr>
<td>$b \leftarrow {0, 1};$</td>
<td>$r' \leftarrow \mathcal{I}(pk, f_{pk}(r));$</td>
</tr>
<tr>
<td>$r \leftarrow {0, 1}^n; s \leftarrow H(r);$</td>
<td>$\text{return}(r = r');$</td>
</tr>
<tr>
<td>$c^* \leftarrow f_{pk}(r) \parallel m_b \oplus s;$</td>
<td></td>
</tr>
<tr>
<td>$b' \leftarrow \mathcal{A}_2^H(c^*);$</td>
<td></td>
</tr>
<tr>
<td>$\text{return}(b = b');$</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.1: IND-CPA experiment for BR encryption scheme and OW assumption.
complicated cryptographic protocols such as the ones presented in Chapter 3 have involved security models where the adversary has access to a plethora of corruption oracles and control over the participants involved in the protocol.

In a similar spirit, the intuition that the one-way trapdoor permutation is hard to invert is expressed in terms of an inverter that receives a challenge—the application of the function to a randomly sampled bitstream—and has to guess the input without access to the secret key.

Figure 1.1 depicts the IND-CPA and the OW experiment, as they are normally defined in code-based cryptography proofs. We work in the random oracle model [29], i.e. hash functions are modeled as truly random functions. The adversary has access to a modified version of the hash oracle that keeps track of its queries in \( L_H \). A proof of IND-CPA security of BR establishes that for any adversary \((\mathcal{A}_1, \mathcal{A}_2)\), \( \Pr[\text{IND-CPA}(\mathcal{A}_1, \mathcal{A}_2) : \text{res}] - \frac{1}{2} \) is “small”. This intuitively captures that an adversary cannot do much better than guessing the secret bit. The notion of small, will be tied to the probability of inverting the trapdoor permutation. In order to relate these probabilities, one has to exhibit a reduction from the IND-CPA security experiment to the OW assumption. This is performed in a stepwise fashion. The sequence of games and the reduction are depicted in Figure 1.2.

In the first transition, we replace the call to the hash function by a randomly sampled value. The distributions of the ciphertext are coincide, except if the adversary manages to query the random oracle with the original input value \( r \). Therefore, we have

\[
\Pr[\text{IND-CPA}(\mathcal{A}_1, \mathcal{A}_2) : \text{res}] \leq \Pr[G_1(\mathcal{A}_1, \mathcal{A}_2) : \text{res}] + \Pr[G_1(\mathcal{A}_1, \mathcal{A}_2) : r \in L_H].
\]

In the next transition, we observe that \( s \) and \( m_b \oplus s \) have the same distribution, therefore the adversary cannot distinguish the challenges received in \( G_1 \) and \( G_2 \). Hence,

\[
\Pr[\text{IND-CPA}(\mathcal{A}_1, \mathcal{A}_2) : \text{res}] \leq \Pr[G_2(\mathcal{A}_1, \mathcal{A}_2) : \text{res}] + \Pr[G_2(\mathcal{A}_1, \mathcal{A}_2) : r \in L_H].
\]

At this point, the challenge does not depend on the secret bit \( b \) any longer, and the best...
the adversary can do is guess. Therefore,

$$\Pr[\text{IND-CPA}(A_1, A_2) : \text{res}] \leq \frac{1}{2} + \Pr[G_2(A_1, A_2) : r \in L_H].$$

Note that we are interested in bounding the probability of the adversary querying value $r$ to the random oracle, with $r$ only occurring in the ciphertext as $f_{pk}(r)$. This means that, intuitively, the adversary manages to invert the one-way trapdoor permutation. The idea is that the inverter $I$ simulates the same environment for the adversary $A$ as the IND-CPA experiment and places the challenge received in the correct position of the ciphertext. Then it explores the list of oracle queries performed by $A$, to find one that corresponds to the preimage of the challenge $y$. Therefore, we have

$$\Pr[G_2(A_1, A_2) : r \in L_H] = \Pr[\text{OW}(I_{BR}(A_1, A_2)) : \text{res}]$$

and putting everything together, after some algebraic manipulation we conclude

$$|\Pr[\text{IND-CPA}(A_1, A_2) : \text{res}] - \frac{1}{2}| \leq \Pr[\text{OW}(I_{BR}(A_1, A_2)) : \text{res}].$$

For this inequality to be meaningful we need to show that the inverter $I_{BR}$ is in the same complexity class as $A$. Being overly informal, the intuition here is that the only overhead introduced by the inverter comes from traversing the set of queries and applying the trapdoor function. Therefore, the complexity of the inverter is a function of the number of queries performed by the adversary and the time it takes for $A_1$ and $A_2$ to run.

A cryptographer would probably be satisfied with the kind of argument outlined above, because the proof steps applied correspond to what he expects and matches his intuition well. However, in EasyCrypt each of these inequalities has to be justified using a combination of program logics. One of these logics, pRHL (probabilistic relational Hoare logic, requires annotations in the form of relational invariants—essentially, assertions that predicate over the states of two programs (rather than one). For the sake of illustrating this gap in the level of reasoning, and without diving into the full details of these logics, the invariants that have to be provided to the tool for the first transition of our sequence are the following:

- $H\{1\} = H\{2\} \land L_H\{1\} = L_H\{2\} \land (\forall x, x \in L_H \iff x \in H)$
- $r \notin L_H \Rightarrow H\{1\} = \{r\} H\{2\}$

for the first and the second calls to the adversary, respectively. At this point, there are a few interesting things to observe.

First, it is clear that this kind of reasoning in terms of the values of variables in the states of the games is much more low-level than what cryptographers are used to, which is to think in terms of probability distributions.

Second, this kind of proofs carry a lot of repetition: if you prove a different scheme you will often need to do a similar step where a hash oracle is eliminated from the encryption function, and the invariants are going to be fairly similar. Modularity plays a crucial role in abstracting away from details of transformations and providing a framework that enables
Chapter 1. Introduction

generic proofs that can be proved once and for all and instantiated many times in the context of different constructions.

Finally, there is a certain structure in the order of the transformations. The overall goal is to make the challenge independent of the secret bit. To achieve this, the ciphertext has to be independent of the message. The only way to make something “disappear” from the ciphertext is to hide it by performing an exclusive-or operation with something randomly distributed. With a given probability loss, a hash can be regarded as randomly distributed. This set of high level principles that appear pervasively in proofs of IND-CPA and IND-CCA security can be abstracted neatly in a small yet powerful logic with pleasing consequences: they are far more accessible to cryptographers; they provide a natural way to structure such proofs and more importantly, they enable automation.

1.3 Challenges

While the work in EasyCrypt was received with praise by the cryptography community, the adoption of the tool has somehow remained elusive and mostly confined to a small group of collaborators. We believe that the problem that has hindered the adoption of our tool in a more widespread way has been a two-fold:

– Overhead: developing machine-checked proofs in any domain poses an overhead over a semi-formal pen and paper argument. In the setting of cryptography proofs formalized in EasyCrypt, probability (in)equalities on each hop between two games are justified using a probabilistic relational Hoare logic. This boils down to writing an invariant that relates the states of the two games, and show that the set of oracles provided in the two games preserves this invariant. The size of the state and the amount of oracles therefore determines how much burden this represents. The amount of games determines how many times this exercise has to be performed. Direct bounds on probabilities are proved using a probabilistic Hoare logic and require many annotations of a different kind.

– Learning Curve: the background required to complete proofs with the tool is usually very different than the one a cryptographer has. In particular, the relational invariants required for completing proofs using the relational Hoare logic may become arbitrarily complex and they are hard to come up with. This impedance mismatch problem piles up on top of the already unexpected overhead described above. To make matters worse, there are some peculiarities of these logics inherited from the probabilistic setting that are somewhat counter-intuitive—even if one has a programming languages or verification background.

Up to this point, we have discussed how to obtain high assurance proofs in the domain of cryptography, by resorting to machine-assisted methods to establish security of

\[ \text{8} \]
1.4. Contributions

our constructions, in the spirit of what Halevi proposed \cite{Halevi}. A natural next step is to aim at building methods that assist in the design of constructions—rather in its a-posteriori analysis. Cryptographic design needs to take into consideration several efficiency factors such as power, time and bandwidth in addition to security. These requirements are extremely important in many application domains. Moreover, multiple trade-offs have to be accommodated for different use cases. Manual approaches are not suitable for achieving resource-aware constructions.

1.4 Contributions

In this thesis, we explore two complementary approaches that can mitigate the overhead of machine-checked cryptography proofs: fully automated analysis and modularity in machine-checked proofs. We apply these two approaches to two different domains: padding-based encryption and one-round key exchange protocols.

Fully Automated Analysis and Synthesis  We propose a novel combination of symbolic and computational methods to achieve automated analysis of classes of constructions, without sacrificing the rigor of the code-based approach, and enabling for the first time the use of program synthesis in cryptographic design with computational guarantees.

The key of our approach is to isolate a set of high-level principles that can be captured using proof rules with a symbolic flavor, and that are sufficient for establishing the security of many examples within a specific class of cryptographic constructions. We believe that such sets of high-level principles exist for many classes of cryptographic constructions, as confirmed by subsequent papers building on this idea (see related work). Moreover, these high level principles tend to be much more familiar and accessible to a cryptographer than the intricacies of program logics.

These proof rules are in turn proved sound with respect to a code-based interpretation. A pleasing consequence is that this soundness proof yields a direct method for transforming derivations in the logic into game-based proofs. Using tools as EasyCrypt as a “certifying back-end” for the logics constitutes mainly an engineering effort.

A converse goal to proving security is finding attacks. Automating attack finding can provide valuable feedback during cryptographic design and can be used to evaluate empirically the completeness of automated security analyses: how many constructions that may be secure cannot be proven so. In this work, we propose fully automated methods for finding attacks against chosen-plaintext and chosen-ciphertext security that hinge on the same symbolic tools that we use for the proofs.

For the sake of concreteness, the work presented here focuses on developing and validating our approach on the simple but practically relevant class of padding-based encryption schemes, i.e. public-key encryption schemes built from hash functions and trapdoor permutations, such as OAEP \cite{Bellare+1994}, OAEP+ \cite{Unruh}, SAEP \cite{Bellare+1999}, or PSS-E \cite{Bellare+2001}.

We propose a novel application of our techniques: automated synthesis of cryptographic constructions. Padding-based encryption schemes are a natural target for automated syn-
thesis, since their design space is relatively small. Proof and attack finding complement each other nicely in this setting. On the one hand, our attack finding techniques yield an automated and efficient means for early pruning of significant portions of the design space. On the other hand, our proof rules enable to automatically obtain proofs of security with concrete bounds of most of the remaining constructions. This pipeline of tools, coined Zoocrypt, connected to a scheme generation algorithm enables us to effectively synthesize and analyze the security of over a million constructions.

Finally, we would like to stress that our work on synthesis of padding based encryption schemes has opened the door for the application of this technique in cryptography and has inspired a stream of work on different domains [11, 81, 12, 96, 65], providing evidence that machine-assisted design of cryptographic constructions can be realised by resorting to relatively simple ideas from program synthesis.

Modular Machine-Checked Proofs There is a vast bulk of work in the cryptographic community devoted to developing generic proofs that apply to a whole class of constructions. One popular class of generic results in cryptography are protocol transformations. If a protocol $\Pi$ is secure with respect to an adversary model $\mathcal{M}$, then $\Pi$ can be transformed into a (more complicated) protocol $\Pi'$ that is secure with respect to a stronger adversary model $\mathcal{M}'$. A nice consequence of this technique is that the simplified analysis in model $\mathcal{M}$ simplifies exploring alternative proofs under different assumptions—a measure for the quality of a cryptography proof is whether the underlying assumptions used are standard.

For key exchange protocols, this approach was pioneered by Bellare, Canetti, and Krawczyk [26] and other transformations have been proposed by Kudla and Paterson [73], Cremers and Feltz [53], and Boyd et al. [40]. This kind of result constitutes a natural target for machine-checked proofs, since the cost overhead of building such proof is justified by its multiple applications. However, EasyCrypt had not been used to obtain machine-checked proofs of neither modular results nor modern key exchange protocols.

We develop a new generic proof of security for key-exchange protocols, and instantiate it to obtain security proofs for known protocols with respect to different adversary models and hardness assumptions. In the cases of Naxos and Nets, we show that it is possible to obtain a CDH proof (without a GAP oracle) if static keys are honestly generated. We also formalize our generic proof and its instantiations using EasyCrypt. This constitutes the largest case study developed using the tool to date.

Structure of the document In Chapter 2, we present our work on automated analysis of Padding-based Encryption Schemes in the Computational Model. Chapter 3 is concerned with our work on Modular Machine-Checked Proofs of One-Round Key Exchange Protocols. In Chapter 4 we review related work. In Chapter 5, we propose directions for future work and present our concluding remarks.
Two different models for analyzing the security of cryptographic constructions have developed over the years and coexist until today, each with its own advantages and disadvantages: the computational model and the symbolic model.

The computational model has its roots in the work of [62] and views cryptographic primitives as functions on bitstrings. Adversaries are modeled as probabilistic algorithms and security is defined in terms of their success probability and computational resources. Security proofs are reductions, showing how a successful attack can be converted into an efficient algorithm that breaks the security of primitives. Often, computational proofs quantify the efficiency of reductions, giving concrete security bounds.

The symbolic model, originating from the seminal work of [56], views cryptographic primitives as function symbols in a term algebra. The properties of primitives and the capabilities of adversaries are modeled using equational theories. This enables automated analysis, but can miss attacks that are possible in the computational model.

A celebrated article by [2] bridges the gap between the two models: it gives sufficient conditions under which symbolic security implies computational security; symbolic methods that exhibit this property are called cryptographically sound. This result, originally proved for symmetric encryption, has since been extended to richer theories [49]. Cryptographic soundness opens the perspective of combining the best of both worlds: fully automated proofs of computational security. However, its applications have remained confined to protocols. Developing cryptographically sound symbolic methods for primitives remains a challenge.

A recent alternative to cryptographically sound symbolic methods uses programming language techniques for reasoning about the security of constructions directly in the computational model. This alternative is embodied in tools like CryptoVerif [35] and EasyCrypt [15], which have been used to verify emblematic cryptographic constructions. Proofs built using these tools yield reductions with concrete security bounds. However, these
tools are primarily interactive and their use requires expert knowledge. Combining them with fully automated methods that apply to large classes of cryptographic primitives has remained an open problem.

A converse goal to proving security is finding attacks. Recent work [7] explores the use of symbolic methods to find attacks and estimate their success probability. Automating attack finding can provide valuable feedback during cryptographic design and be used to evaluate empirically the completeness of automated security analyses by answering questions such as how many constructions that may be secure cannot be proven so.

**Contributions**  We present new methods for automatically analyzing the security of padding-based encryption schemes, i.e. public-key encryption schemes built from hash functions and trapdoor permutations, such as OAEP [30], SAEP [39] or PSS-E [48]. Collectively, the methods presented can report a proof, with a concrete security bound, or an attack. These methods rest on two main technical contributions.

First, we introduce specialized logics for proving chosen-plaintext and chosen-ciphertext security. Proof rules have a symbolic flavor and proof search can be automated, yet one can extract concrete security bounds from valid derivations. This is achieved through a novel combination of symbolic and computational methods; for instance, the logics use symbolic deducibility relations for computing bounds of the probability of events, and for checking the existence of reductions to computational assumptions.

Second, we propose fully automated methods for finding attacks against chosen-plaintext and chosen-ciphertext security. Our methods are inspired by static equivalence [1], and exploit algebraic properties of trapdoor permutations to find attacks against realizations of schemes that are consistent with computational assumptions.

We demonstrate the strengths of our methods by implementing a toolset, called ZooCrypt, for analyzing fully automatically a set of user-given or machine-generated schemes. We generate more than one million examples and use the toolset to analyze their (in)security. The output of the analysis is a database that records for each scheme and set of assumptions, either an adversary that breaks the security of the scheme or a formal derivation that proves its security.

We stress that we focus on padding-based encryption primarily for illustrative purposes. Padding-based schemes include many practically relevant constructions, and provide an excellent testbed to evaluate the effectiveness of our methods and illustrate their working.

**Contents**  §2.1 provides background on public-key encryption schemes and their security; §2.2 introduces symbolic tools that are the base of our analyses; §2.3 and §2.4 describe logics for chosen-plaintext and chosen-ciphertext security respectively, while §2.5 covers attack finding techniques; §2.6 reports on the implementation of ZooCrypt and on its evaluation.
2.1. Cryptographic Background

This section provides some background on padding-based public-key encryption schemes and their security.

2.1.1 Padding-Based Public-Key Encryption

A public-key encryption scheme is a triple of probabilistic algorithms \((KG, E, D)\):

Key Generation The key generation algorithm \(KG\) outputs a pair of keys \((pk, sk)\); \(pk\) is a public-key used for encryption, \(sk\) is a secret-key used for decryption;

Encryption Given a public-key \(pk\) and a message \(m\), \(E_{pk}(m)\) outputs a ciphertext \(c\);

Decryption Given a secret-key \(sk\) and a ciphertext \(c\), \(D_{sk}(c)\) outputs either message \(m\) or a distinguished value \(\perp\) denoting failure.

We require that for pairs of keys \((pk, sk)\) generated by \(KG\), \(D_{sk}(E_{pk}(m)) = m\) holds for any message \(m\).

We study public-key encryption schemes built from trapdoor permutations, hash functions and basic bitstring operations, such as bitwise exclusive-or \((\oplus)\), concatenation \((\|)\) and projection. We let \(\{0, 1\}^n\) denote the set of bitstrings of length \(n\) and \([x]_n^\ell\) the bitstring obtained from \(x\) by dropping its \(n\) most significant bits and taking the \(\ell\) most significant bits of the result.

Trapdoor permutations are functions that are easy to evaluate, but hard to invert without knowing a secret trapdoor. Formally, a trapdoor permutation scheme on \(\{0, 1\}^n\) consists of a probabilistic permutation generator \(PG\) that returns permutations \(f\) and \(f^{-1}\) on \(\{0, 1\}^n\) that are inverse of each other. The security of a trapdoor permutation scheme is measured by the probability that an adversary (partially) inverts \(f\) on a random input.

**Definition 2.1 (One-Way Trapdoor Permutation).** Let \(PG\) be the permutation generator of a trapdoor permutation scheme on \(\{0, 1\}^n\) and let \(k, \ell\) be such that \(k + \ell \leq n\). The probability of an algorithm \(I\) in \(q\)-inverting \(PG\) on bits \([k \ldots k + \ell - 1]\) is

\[
\text{Succ}^{\text{pd-OW}}_{k, \ell, q, t}(I) \overset{\text{def}}{=} \Pr[\text{OW}(I) : [s]_k^\ell \in S \land |S| \leq q]
\]

where \(\text{OW}(I)\) is the experiment:

\[(f, f^{-1}) \leftarrow PG; s \overset{\$}{\leftarrow} \{0, 1\}^n; S \leftarrow I(f(s), f)\]

This definition generalizes the set partial-domain one-wayness problem of [58], which corresponds to the case \(k = 0\). We define \(\text{Succ}^{\text{pd-OW}}_{k, \ell, q, t}(k, \ell)\) as the maximal success probability over all inverters executing in time \(t\), and omit the first parameter when it is 0. When \(k = 0, \ell = n,\) and \(q = 1\), the experiment corresponds to the standard one-wayness problem, and we note this quantity \(\text{Succ}^{\text{OW}}_{\ell, q}(t)\).
2.1.2 Security of Padding-Based Encryption Schemes

We consider two security notions for public-key encryption: chosen-plaintext security, or IND-CPA, and adaptive chosen-ciphertext security, or IND-CCA. Both can be described succinctly using the following experiment, or game, in which an adversary \( A \) represented as a pair of procedures \((A_1, A_2)\) interacts with a challenger:

\[
\begin{align*}
(pk, sk) & \leftarrow KG; \\
(m_0, m_1, \sigma) & \leftarrow A_1^H(pk); \\
b & \leftarrow \{0, 1\}; \\
c^* & \leftarrow E(m_b); \\
\bar{b} & \leftarrow A_2^H(c^*, \sigma); \\
\text{return } b = \bar{b}
\end{align*}
\]

The challenger first generates a fresh pair of keys \((pk, sk)\) and gives the public key \(pk\) to the adversary, which returns a pair of messages \((m_0, m_1)\) of its choice. The challenger then samples uniformly a bit \(b\), encrypts the message \(m_b\) and gives the resulting ciphertext \(c^*\) (the challenge) to the adversary. Finally, the adversary outputs a guess \(\bar{b}\) for \(b\). Note that \(A_1\) and \(A_2\) can communicate with each other through \(\sigma\) and have oracle access to all hash functions \(\vec{H}\) used in the scheme. We call this basic experiment CPA; we define the experiment CCA similarly, except that the adversary \(A\) is given access to a decryption oracle \(D\), with the proviso that \(A_2\) cannot ask for the decryption of the challenge ciphertext \(c^*\). In the remainder, we note \(L_H\) the list of queries that the adversary makes to a hash oracle \(H\) in either experiment, and \(L_D\) the list of decryption queries that \(A_2\) makes during the second phase of the CCA experiment.

Security is measured by the advantage of an adversary playing against a CPA or CCA challenger.

**Definition 2.2 (Adversary Advantage).** The CPA-advantage and the CCA-advantage of an encryption scheme \(\Pi = (KG, E, D)\) is defined respectively as:

\[
\begin{align*}
\text{Adv}^{\text{CPA}}(\Pi)(A) & \overset{\text{def}}{=} 2 \Pr[\text{CPA}_\Pi(A) = \text{true}] - 1 \\
\text{Adv}^{\text{CCA}}(\Pi)(A) & \overset{\text{def}}{=} 2 \Pr[\text{CCA}_\Pi(A) = \text{true}] - 1
\end{align*}
\]

We define \(\text{Adv}^{\text{CPA}}_\Pi(t, \vec{q})\) (resp. \(\text{Adv}^{\text{CCA}}_\Pi(t, \vec{q})\)) as the maximal advantage over all adversaries \(A\) that execute within time \(t\) and make at most \(q_O\) queries to oracle \(O\).

To define asymptotic security, we consider instead of a single scheme, a family of schemes \(\Pi_\eta\) indexed by a security parameter \(\eta \in \mathbb{N}\). The above probability quantities thus become functions of \(\eta\). We say that a scheme is secure if the advantage of any adversary executing in time polynomial in \(\eta\) is a negligible function \(\nu\) of \(\eta\), i.e. \(\forall c. \exists n_c. \forall n > n_c. \nu(n) < n^{-c}\).

A security proof for a padding-based encryption scheme is a reduction showing that any efficient adversary against the security of the scheme can be turned into an efficient
2.2. Symbolic Tools

inverter for the underlying trapdoor permutation. Security proofs are typically in the random oracle model, where hash functions are modeled as truly random functions. One seemingly artificial property of the random oracle model is that reductions to computational assumptions are allowed to adaptively choose the responses of oracles modeling hash functions; this property is called *programmability*. In contrast, we only consider proofs in the non-programmable random oracle model, where reductions are only allowed to observe oracle queries and responses.

For illustrative purposes, we use OAEP [30] as a running example. RSA-OAEP, which instantiates OAEP with RSA as trapdoor permutation is recommended by several international standards.

**Definition 2.3** (OAEP). Let \( \mathcal{PG} \) be permutation generator for a trapdoor permutation scheme on \( \{0,1\}^n, \rho \in \mathbb{N} \), and let

\[
G : \{0,1\}^k \rightarrow \{0,1\}^\ell \\
H : \{0,1\}^\ell \rightarrow \{0,1\}^k
\]

be two hash functions such that \( n = k + \ell \) and \( \rho < \ell \). OAEP is composed of the following triple of algorithms:

\[
\begin{align*}
\mathcal{KG} & \overset{\text{def}}{=} (f, f^{-1}) \leftarrow \mathcal{PG}; \quad \text{return} \ (f, f^{-1}) \\
\mathcal{E}_f(m) & \overset{\text{def}}{=} r \leftarrow \{0,1\}^k; \quad s \leftarrow G(r) \oplus (m\|0^\rho); \quad t \leftarrow H(s) \oplus r; \quad \text{return} \ f(s\|t) \\
\mathcal{D}_{f^{-1}}(c) & \overset{\text{def}}{=} s\|t \leftarrow f^{-1}(c); \quad r \leftarrow t \oplus H(s); \quad m \leftarrow s \oplus G(r); \\
& \quad \text{if} \ [m]_{\rho}^{\ell-\rho} = 0^\rho \ \text{then return} \ [m]_{0}^{\ell-\rho} \ \text{else return} \ \perp
\end{align*}
\]

2.2 Symbolic Tools

We introduce algebraic expressions to model padding-based encryption schemes and symbolic notions to reason about equality and computability of expressions.

2.2.1 Syntax and Semantics of Expressions

Expressions are built from a set \( \mathcal{R} \) of random variables and a set \( \mathcal{X} \) of variables, using bitstring operations, and hash function symbols drawn from a set \( \mathcal{H} \).\(^1\) The grammar of

\(^1\)For ease of presentation, we restrict ourselves to the case where a single one-way permutation is used. Our results directly generalize to the setting of multiple one-way permutations and our toolchain supports this more general setting.
expressions is

\[
\begin{align*}
e &::= x & \text{variable} \\
| & r & \text{random variable} \\
| & 0^n & \text{all-zero bitstring of length } n \\
| & f(e) & \text{permutation} \\
| & f^{-1}(e) & \text{inverse permutation} \\
| & H(e) & \text{hash function} \\
| & e \oplus e & \text{exclusive-or} \\
| & e \parallel e & \text{concatenation} \\
| & [e]_n^\ell & \text{drop } n \text{ bits then take } \ell \text{ bits}
\end{align*}
\]

where \( x \in X, r \in R, n, \ell \in S, \) and \( H \in \mathcal{H} \). To avoid unnecessary parentheses, we let \( \oplus \) take precedence over concatenation. For instance, the encryption algorithm of OAEP may be represented as the expression:

\[
f(G(r) \oplus (m \parallel 0) \parallel H(G(r) \oplus (m \parallel 0)) \oplus r).
\]

Expressions are typed using a size-based type system. A type \( \ell \in S \) is either a size variable or the sum \( k + n \) of two types. We note \( |e| \) the type of an expression \( e \), which intuitively represents the length of the bitstring it denotes. For instance, the type of \( e_1 \parallel e_2 \) is \( |e_1| + |e_2| \) and \( e_1 \oplus e_2 \) is well-typed if \( |e_1| = |e_2| \). In an asymptotic setting, we interpret size variables as functions that grow polynomially with the security parameter.

We let \( R(e) \), \( X(e) \) and \( \mathcal{H}(e) \) denote, respectively, the sets of random variables, variables, and hash function symbols occurring in an expression \( e \). We note \( e \{ e_1 / e_0 \} \) the simultaneous substitution of \( e_0 \) by \( e_1 \) in \( e \) and \( ST_H(e) \) the set of sub-expressions of \( e \) that are of the form \( H(e') \). Abusing notation, we write \( e \in R(e_1, e_2) \) instead of \( R(e_1) \cup R(e_2) \).

We define a code-based interpretation \( \langle e \rangle_C \) that translates an algebraic expression into an expression in the programming language used to specify games (see [95, 31, 15]). Here, the variables in \( R(e) \) and \( X(e) \), the hash function symbols in \( \mathcal{H} \), and \( f \) and \( f^{-1} \) are interpreted as program variables. The remaining function symbols are interpreted in the standard way. Concretely, we define \( \langle x \rangle_C = x \) for \( x \in X \cup R \), \( \langle e \parallel e' \rangle_C = \langle e \rangle_C \parallel \langle e' \rangle_C \), \( \langle f(e) \rangle_C = f(\langle e \rangle_C) \), \( \langle H(e) \rangle_C = H(\langle e \rangle_C) \), and so on. We always assume that all functions and variables used by \( e \) are defined in the context where \( \langle e \rangle_C \) is used.

Note that this (deterministic) interpretation of expressions does not distinguish between variables and random variables, but we will later use the distinction to determine which values must be sampled before evaluating \( \langle e \rangle_C \).

### 2.2.2 Equality and Deducibility

We model algebraic properties of bitstrings using the equational theory \( =_E \), defined as the smallest congruence relation on well-typed expressions that contains all instances of the axioms of Figure 2.1. The relation \( =_E \) is sound, i.e., for all expressions \( e, e' \) and commands \( c_1, c_2, e =_E e' \) implies that the programs \( c_1; x \leftarrow \langle e \rangle_C; c_2 \) and \( c_1; x \leftarrow \langle e' \rangle_C; c_2 \) are equivalent.
2.3. Chosen-Plaintext Security

\[(e_1 \oplus e_2) \oplus e_3 = e_1 \oplus (e_2 \oplus e_3) \quad e_1 \oplus e_2 = e_2 \oplus e_1 \quad e \oplus 0 \equiv e \quad e \oplus e = 0 \equiv e\]

\[0^n_k \equiv 0^k \quad (e_1 \parallel e_2) \parallel e_3 = e_1 \parallel (e_2 \parallel e_3) \quad [e_1 \parallel e_2]_{e_1} = e_1 \quad [e_1 \parallel e_2]_{[e_1]} = [e_2]_{e_1}\]

\[e_1 \oplus e_2]_n = [e_1]_n \oplus [e_2]_n \quad [e]_{[e]} = e \quad [e]_n \parallel [e]_{n+\ell} = [e]_{n+\ell}\]

\[f(f^{-1}(e)) = e \quad f^{-1}(f(e)) = e\]

Figure 2.1: Equational theory for algebraic expressions

We use the standard notion of deducibility from symbolic cryptography to reason about computability. Informally, an expression \(e'\) is deducible from \(e\) if there exists an efficient algorithm that computes \(e'\) from \(e\). We describe such algorithms symbolically using contexts. A context is a symbolic expression with no variables except for (possibly) a distinguished variable \(\Box\).

**Definition 2.4 (Deducibility).** An expression \(e'\) is deducible from another expression \(e\) using a context \(C\), written \(e \vdash_C e'\), if \(C\) is an expression such that

i) \(\mathcal{R}(C) = \emptyset\), \(\mathcal{X}(C) \subseteq \{\Box\}\);

ii) \(f^{-1}\) does not occur in \(C\), and

iii) \(C\{e/\Box\} =_{E} e'\).

We write \(e \not\vdash e'\) if there exists a context \(C\) such that \(e \not\vdash_C e'\). We define \(\not\vdash^*\) analogously, but dropping restriction ii).

It is not hard to see that \(\vdash\) is sound in the following sense: \(e \vdash e'\) implies that there exists an efficient algorithm that computes \(e'\) from \(e\) given \(f\). This algorithm can be obtained from \(C\), the context witnessing \(e \vdash e'\). We obtain an identical result for \(\vdash^*\), except that the corresponding algorithm is also given \(f^{-1}\). Note that we do not require \(\not\vdash\)-completeness, also known as \(\not\vdash\)-soundness [25], since we never use the deducibility relation to conclude that an expression is not computable from another.

### 2.3 Chosen-Plaintext Security

This section introduces a logic for proving chosen-plaintext security of padding-based encryption schemes by reduction to computational assumptions on the underlying trapdoor permutations. For the sake of readability, our presentation separates the derivation of valid judgments from the computation of concrete security bounds from valid derivations.
Chapter 2. Automated analysis of padding-based encryption

| L1 | sample function $H \in \mathcal{H}$ | $r \notin \{0,1\}^k$; $H \notin \{(0,1)^k \rightarrow \{0,1\}^k\}$; $G \notin \{(0,1)^k \rightarrow \{0,1\}^k\}$; |
| L2 | $(f, f^{-1}) \rightarrow \mathcal{P}G$; | $(f, f^{-1}) \leftrightarrow \mathcal{P}G$; |
| L3 | $(m_0, m_1, \sigma) \leftrightarrow A_1^R(f)$; | $(m_0, m_1, \sigma) \leftrightarrow A_1^{H,G}(f)$; |
| L4 | $b \notin \{0,1\}$; $m \leftarrow m_b$; | $b \notin \{0,1\}$; $m \leftarrow m_b$; |
| L5 | sample bitstring $r \in \mathcal{R}(c^*, \phi)$; | $\breve{b} \leftarrow A_2^G(f(G(r) \oplus (m \parallel 0) \parallel H(G(r) \oplus (m \parallel 0)) \oplus r))$; |
| L6 | $\breve{b} \leftarrow A_2^G(f(G(r) \oplus (m \parallel 0) \parallel H(G(r) \oplus (m \parallel 0)) \oplus r))$; | $\breve{b} = \breve{b}$ |
| L7 | return $\{\phi\}_C$ | return $\{\phi\}_C$ |

Figure 2.2: Generic CPA game for $c^*: \phi$ on the right and instantiated CPA game for initial OAEP judgment $f(G(r) \oplus (m \parallel 0) \parallel H(G(r) \oplus (m \parallel 0)) \oplus r)$ : Guess on the left.

2.3.1 Judgments

Judgments predicate over the probability of an event $\phi$ in the CPA experiment where the challenge ciphertext is computed using $(c^*)_C$. Events are drawn from the grammar

$$\phi ::= \text{Guess} | \text{Ask}(H_1,e_1) \land \ldots \land \text{Ask}(H_k,e_k) \quad \text{where } k \geq 1$$

where Guess corresponds to the adversary correctly guessing the hidden bit $b$, and Ask($H,e$) corresponds to the adversary querying $e$ to $H$.

Judgments are of the form $\vdash_{P^G} c^*: \phi$, where $c^*$ is a well-typed expression with $\mathcal{X}(c^*) \subseteq \{m\}$, $\phi$ is an event, and $\breve{p} \in \{0,1/2\}$ is a probability tag. For a one-way permutation with permutation generator $\mathcal{P}G$ and an adversary $\mathcal{A}$, we associate to each judgment the probability

$$\Pr[c^*: \phi]_{\mathcal{P}G,\mathcal{A}} = \Pr[\text{CPA}_{\mathcal{P}G,\mathcal{A}}(c^*, \phi) = \text{true}]$$

where CPA$_{\mathcal{P}G,\mathcal{A}}(c^*, \phi)$ is defined in Figure 2.2.

We also define $\Pr[c^*: \breve{\phi}]_{\mathcal{P}G,\mathcal{A},\breve{q}}$ as the maximum of $\Pr[c^*: \phi]_{\mathcal{P}G,\mathcal{A}}$ over all adversaries executing within time $t_A$ and performing at most $\breve{q}$ queries. The CPA logic is designed to ensure that if $\vdash_{P^G} c^*: \phi$ is derivable, then the probability $\Pr[c^*: \phi]_{\mathcal{P}G,\mathcal{A},\breve{q}}$ is upper bounded by $\breve{p}$ plus a negligible term. In particular, if $\vdash_{1/2} c^*: \text{Guess}$ is derivable, then any scheme II with encryption algorithm $(c^*)$ is asymptotically secure against chosen-plaintext attacks (see § 2.3.4).

Derivability in the logic is parametrized by a set $\Gamma$ of computational assumptions of the form $(k, \ell)$ on $f$, where $k, \ell \in \mathcal{S}$. Such assumptions state that it is hard to compute $[r]_k^\ell$ from $f(r)$.

2.3.2 Rules

Figure 2.3 presents the rules of our logic. A proof in our logic proof proceeds by transforming the ciphertext in the judgment until it is independent of the message. This can be

---

2Probability tags increase the readability of rules and support a more uniform interpretation of judgments. However, they can be deduced from the shape of the event: for every valid judgment $\vdash_{P^G} c^*: \phi$, the probability tag $\breve{p}$ is $1/2$ iff $\phi = \text{Guess}$. 

---
2.3. Chosen-Plaintext Security

Bridging Rules:

\[
\begin{align*}
\text{[Opt]} & \quad \frac{\vdash_{\phi} \{c^* : \phi \} \{e \oplus r/r\}}{\vdash_{\phi} c^* : \phi} \\
\text{[Sub]} & \quad \frac{\vdash_{\phi} c_1^* : \phi_1, \ c_1^* = e \ c_2^* \land \phi_1 \Longrightarrow e \ \phi_2}{\vdash_{\phi} c_2^* : \phi_2}
\end{align*}
\]

\[
\begin{align*}
\text{[Perm]} & \quad \frac{\vdash_{\phi} \{c^* : \phi \} \{\vec{r} / f(\vec{r})\}}{\vdash_{\phi} c^* : \phi} \\
\text{dist}(\vec{r}) \land \quad r_i \notin \mathcal{R}((c^* : \phi) \{0 / f(\vec{r})\})
\end{align*}
\]

\[
\begin{align*}
\text{[Split]} & \quad \frac{\vdash_{\phi} \{c^* : \phi \} \{r_1 \parallel r_2 / r\}}{\vdash_{\phi} c^* : \phi} \\
\quad r_1 \neq r_2 \land \quad r_1, r_2 \notin \mathcal{R}(c^*, \phi)
\end{align*}
\]

Hash Rules:

\[
\begin{align*}
\text{[Hash]} & \quad \frac{\vdash_{\phi} \{c^* : \phi \} \{r / H(e)\}}{\vdash_{\phi} c^* : \phi} \\
\text{ST}_{H}(c^*, \phi) = \{H(e)\} \land \quad r \notin \mathcal{R}(c^*, \phi)
\end{align*}
\]

\[
\begin{align*}
\text{[Hash]} & \quad \frac{\vdash_{\phi} \{c^* : \phi \} \{r / H(e)\}}{\vdash_{\phi} c^* : \phi} \\
\text{ST}_{H}(c^*, \phi) = \{H(e)\} \land \quad r \notin \mathcal{R}(c^*, \phi)
\end{align*}
\]

Terminal Rules:

\[
\begin{align*}
\text{[Ind]} & \quad \frac{0 \vdash_{\phi} c^* : \text{Guess}}{\vdash_{\phi, \text{Ind}} c^*} \\
\text{m} \notin \mathcal{X}(c^*)
\end{align*}
\]

\[
\begin{align*}
\text{[Rnd]} & \quad \frac{0 \vdash_{\phi} c^* : \text{Ask}(H_1, e_1) \land \cdots \land \text{Ask}(H_n, e_n)}{\vdash_{\phi} c^*: \text{Guess}} \\
\vec{e} \parallel \mathcal{R}(c^*) \parallel m \vdash_{\phi} \vec{r} \land \quad \vec{r} \subseteq \mathcal{R}(\vec{e}) \setminus \mathcal{R}(c^*) \land \text{dist}(\vec{r})
\end{align*}
\]

\[
\begin{align*}
\text{[OW]} & \quad \frac{0 \vdash_{\phi} c^* : \text{Ask}(H_1, e_1) \land \cdots \land \text{Ask}(H_n, e_n)}{\vdash_{\phi} c^*: \text{Guess}} \\
f(\vec{r}_1) \parallel \vec{r}_2 \parallel m \vdash_{\phi} c^* \land \text{dist}(\vec{r}_1) \land \quad \vec{e} \parallel \vec{r}_2 \parallel m \vdash_{\phi} [\vec{r}_1{k}^e] \land \text{dist}(\vec{r}_2) \land \quad \mathcal{R}(c^*, \vec{e}) = \vec{r}_1 \cup \vec{r}_2 \land (k, \ell) \in \Gamma
\end{align*}
\]

Figure 2.3: Proof rules of the CPA logic (we use dist(\vec{e}) to denote that \vec{e} consists of distinct expressions and \vec{e} to denote the concatenation of its elements when used as an expression)

achieved using bridging and hash rules. Bridging rules reformulate judgments into equivalent forms. Hash rules formalize the equivalence up to failure principle and exploit that an adversary cannot distinguish between the result of a hash application \(H(e)\) and a random bitstring \(r\) if he never queries \(e\) to \(H\). Corresponding proof obligations, we use terminal rules that capture information-theoretic reasoning and reductions to computational assumptions on the employed one-way permutation.
We decorate the terminal rules [Rnd] and [OW] with information required to compute concrete bounds. (§ 2.3.3). We only allow to apply a rule if all instances of expressions are well-typed; this implies that for any valid derivation \( \triangledown \) there exists a common typing environment for all rule applications in which all expressions involved in the derivation are well-typed.

The rules [Opt], [Perm], and [Split] correspond to bridging steps. The rule [Opt] corresponds to optimistic sampling, which allows substituting \( r \) by \( e \oplus r \) provided that \( r \notin \mathcal{R}(e) \). The rule [Perm] replaces all occurrences of \( f(\vec{r}) \) by \( \vec{r} \) provided that \( \vec{r} \) consists of distinct random variables that are only used as arguments to \( f \). The rule [Split] replaces a random bitstring \( r \) with the the concatenation of two fresh random bitstrings \( r_1 \) and \( r_2 \). The rule [Sub] can be used to weaken the event by removing conjuncts and to replace expressions in judgments with equivalent expressions. This rule is often needed to perform simplification after the application of another bridging rule: e.g. after applying rule [Opt] a judgement that contains \( H(e \oplus (e \oplus r)) \) can be obtained and this can be further simplified to \( H(r) \) by using [Sub].

Both hash rules simplify the original judgment by replacing \( H(e) \) with a fresh random variable \( r \) in both ciphertext \( c \) and event \( \phi \). The resulting game is equivalent to the original game unless the adversary queries \( H(e) \). Both hash rules have an additional premise that bounds this probability. The rule [Hash1] bounds the probability of the event \( \text{Ask}(H, e) \) in the modified game. The rule can be justified by Shoup’s Fundamental Lemma [95]. The rule [Hash2] bounds the probability of \( \phi \land \text{Ask}(H, e) \) in the original game and can be justified by a case distinction on \( \text{Ask}(H, e) \).

Finally, the rules [Ind], [Rnd], and [OW] are terminal and provide a means of directly bounding the probability of an event. The first two rules are information-theoretic, whereas the third formalizes a reduction to a computational assumption. The rule [Ind] closes a branch when the challenge ciphertext no longer depends on the plaintext variable \( m \); in terms of the CPA experiment, this means that the challenge is independent from the hidden bit \( b \), and hence the probability that an adversary guesses right is exactly \( 1/2 \). The rule [Rnd] closes a branch in the case when the adversary manages to make certain oracle queries that require guessing random values that are independent from its view. The rule [OW] closes a branch when it is possible to find a reduction to the problem of partially inverting a trapdoor permutation. The contexts \( C \) and \( C' \) witnessing the derivability conditions allow us to define the following inverter \( I(y, f) \):

1. Call \( A_1(f) \) to obtain \( (m_0, m_1, \sigma) \), sample the bit \( b \), and set \( m = m_b \).
2. Sample \( \vec{r}_2 \) and compute the challenge ciphertext as \( c^* = \langle C'(y \parallel \vec{r}_2 \parallel m / \sigma) \rangle_C \).
3. Call \( A_2((c^*)_C, \sigma) \) and return the set \( \{ \langle C(x_1 \parallel \ldots \parallel x_n \parallel \vec{r}_2 \parallel m / \sigma) \rangle_C \mid x_1 \in L_{H1}, \ldots, x_n \in L_{Hn} \} \).

Since the one-way challenger calls the inverter \( I \) with \( y = f(\vec{r}_1) \), the side conditions of [OW] imply that \( \langle C(\vec{x} \parallel \vec{r}_2 \parallel m / \sigma) \rangle_C = [\vec{r}_1]_{k'}^{-} \) for at least one vector \( \vec{x} \) of values queried to the random oracles.
2.3. Chosen-Plaintext Security

\[ B \left( t_A, \vec{q} \right) (\nabla) = \begin{cases} 
B(\nabla_1) + B(\nabla_2) & \text{if } L_\nabla = [\text{Hash}_{1,2}] \\
1/2 & \text{if } L_\nabla = [\text{Ind}] \\
q_\nabla \times 2^{-\ell} & \text{if } L_\nabla = [\text{Rnd}] \\
\text{Succ}_{\Theta_t}^{s-pd-OW}(k, \ell, q_\nabla, t_I) & \text{if } L_\nabla = [\text{OW}_{1}^k(C, C')] \\
B(\nabla_1) & \text{otherwise}
\end{cases} \]

where \( T(C) \) is the time required to execute the algorithm corresponding to \( C \) and \( q_\nabla = \prod_{i=1}^{n} q_i H_i \).

\( E_\nabla = \text{Ask}(H_1, e_1) \land \ldots \land \text{Ask}(H_n, e_n) \).

Figure 2.4: Computation of concrete security bounds from logical derivations

2.3.3 Bounds

Security bounds \( p \) are drawn from the grammar

\[ p ::= \epsilon \mid 1/2 + \epsilon \quad q ::= 1 \mid q \times q \mid q_H \quad t ::= t_A \mid q \mid q \times t_f \mid t + t \]

\[ \epsilon ::= 0 \mid 2^{-k} \mid \text{Succ}_{\Theta_t}^{s-pd-OW}(k, \ell, q, t) \mid q \times \epsilon \mid \epsilon + \epsilon \]

where \( k, \ell \in \mathcal{S}, H \in \mathcal{H}, f \in \mathcal{F}, t_f \) is a worst-case bound for the time required to evaluate \( f \), and \( t_A \) is the execution time of \( A \).

Security bounds are computed from a derivation \( \nabla \) in the logic using a function \( B \) parametrized by the computational resources of an adversary, namely, its execution time \( t_A \) and the number \( q \) of oracle queries it can make. The definition of function \( B \) is given in Figure 2.4. In the recursive calls, we omit resource parameters since they remain unchanged. Moreover, for a derivation \( \nabla \) with rule \( R \) at its root, we use \( L_\nabla \) to refer to \( R \)'s label, \( E_\nabla \) to refer to the event in \( R \)'s conclusion, and \( \nabla_i \) to refer to the derivation of the \( i \)-th premise of \( R \).

For all bridging rules, the bound is inherited from their single premise. For rules [\text{Hash}_1] and [\text{Hash}_2], the bound is computed as \( p_1 + p_2 \) where \( p_1 \) is the probability of the original event in the transformed experiment and \( p_2 \) bounds the probability of failure. The rule [\text{Ind}] yields an exact bound that can be readily used. The rule [\text{Rnd}] correspond to the probability of guessing a random bitstring of length \( \ell \) in \( q_\nabla \) tries. In the case of rule [\text{OW}], the bound is the maximal success probability against set partial-domain one-wayness over all inverters using the same resources as the reduction we construct. When \( k = 0 \) and \( \ell = |f| \), we can alternatively use the standard reduction from set one-wayness to one-wayness to obtain the bound \( \text{Succ}_{\Theta_t}^{\text{OW}}(t_A + T(C') + q_\nabla \times (T(C) + t_f)) \). Here, the adjusted time bound accounts for the fact that the inverter inspects every combination of queries made to the random oracle queries used by \( C \), computes a value, applies \( f \) to that value and compares the result to its challenge to find the right pre-image.
Chapter 2. Automated analysis of padding-based encryption

2.3.4 Soundness

Let $c^*$ be an expression, $\phi$ an event and $\hat{p}$ a probability tag.

**Theorem 2.1.** If $\nabla$ is a derivation of $\models_{\hat{p}} c^* : \phi$ under $\Gamma$, then

$$\Pr[c^* : \phi]_{PG,t_A,q} \leq B_{(t_A,q)}(\nabla)$$

We defer the proof of this theorem to § 2.7.

**Corollary 2.2.** If $\langle c^* \rangle_{C}$ is the encryption algorithm of an encryption scheme $\Pi$ and $\nabla$ is a derivation of $\models_{1/2} c^* : \text{Guess}$ under $\Gamma$, then

$$\text{Adv}^{\text{CPA}}_{\Pi}(t_A, q) \leq 2B_{(t_A,q)}(\nabla) - 1.$$  

Moreover, this bound is negligible if $t_A$ and all $q_H$ in $\vec{q}$ are polynomial in the security parameter, and all assumptions in $\Gamma$ hold.

**Proof sketch.** Follows from Theorem 2.1 and the observation that

$$\text{Adv}^{\text{CPA}}_{\Pi}(t_A, q) = 2 \Pr[c^* : \text{Guess}]_{PG,t_A,q} - 1$$

since $\langle c^* \rangle_{C}$ is the encryption algorithm of $\Pi$. 

2.3.5 Examples

In this section we give detailed proofs of security for OAEP and PSS-E in the form of derivations in the logics presented in the previous section.

2.3.5.1 OAEP

We illustrate the use of the CPA logic to prove the chosen-plaintext security of OAEP under one-wayness (OW) and partial-domain one-wayness (PDOW) assumptions on the underlying trapdoor permutation. Additionally, we give the concrete security bounds extracted from the corresponding derivations.

The proofs under one-wayness and partial-domain one-wayness share a common part, depicted at the bottom of Figure 2.5. This first part proceeds by applying $[\text{Hash}_1]$ to replace the expression $G(r)$ with a fresh random bitstring $r'$. To close the branch corresponding to the $\text{Guess}$ event, we apply optimistic sampling, replacing $r' \oplus (m \parallel 0)$ by $r'$. In the obtained judgment, the challenge ciphertext does not depend on the plaintext, so we conclude by applying $[\text{Ind}]$. We continue with the branch corresponding to the failure event $\text{Ask}(G, r)$ applying optimistic sampling as for the previous branch. At this point there are two different ways to proceed, that yield proofs w.r.t. different hypotheses.

The derivation for PDOW is depicted in the middle of Figure 2.5. The proof proceeds by applying rule $[\text{Hash}_1]$ to replace $H(r)$ by a fresh random bitstring $r''$. The rule $[\text{Opt}]$ is then applied in both premises to replace $r'' \oplus r$ by just $r$. The branch corresponding to
2.3. Chosen-Plaintext Security

\[ \text{\textbackslash OW} \]

\[
\begin{align*}
\frac{\vdash_0 f(r' \parallel r) : \text{Ask}(G, r') \quad [\text{Rnd}]}{\vdash_0 f(r'' \parallel r) : \text{Ask}(G, r'' \oplus r) \quad [\text{Opt}]}
\end{align*}
\]

\[
\frac{\vdash_0 f(r'' \parallel r) : \text{Ask}(G, H(r') \oplus r) \quad [\text{Opt}]}{\vdash_0 f(r' \parallel (m \parallel 0)) \parallel H(r' \oplus (m \parallel 0)) \oplus r) : \text{Ask}(G, r) \quad [\text{Opt}]}
\]

\[ \text{\textbackslash PDOW} \]

\[
\begin{align*}
\frac{\vdash_0 f(r' \parallel r') : \text{Ask}(G, r) \quad [\text{Rnd}]}{\vdash_0 f(r'' \parallel r' \oplus r) : \text{Ask}(G, r) \quad [\text{Opt}]}
\end{align*}
\]

\[
\frac{\vdash_0 f(r'' \parallel r') : \text{Ask}(H, r') \quad [\text{Opt}]}{\vdash_0 f(r'' \parallel r'' \oplus r) : \text{Ask}(H, r') \quad [\text{Opt}]}
\]

\[
\frac{\vdash_0 f(r'' \parallel r') : \text{Ask}(H, r') \quad [\text{Opt}]}{\vdash_0 f(r'' \parallel r' \oplus r) : \text{Ask}(G, r) \quad [\text{Hash}]}
\]

\[ \text{\textbackslash OAEP} \]

\[
\begin{align*}
\frac{\vdash_{1/2} f(r'' \parallel H(r') \oplus r) : \text{Guess} \quad [\text{Ind}]}{\vdash_{1/2} f(r'' \parallel (m \parallel 0)) \parallel H(r' \oplus (m \parallel 0)) \oplus r) : \text{Guess} \quad [\text{Opt}]}
\end{align*}
\]

\[
\frac{\vdash_{1/2} f((G(r) \oplus (m \parallel 0)) \parallel H(G(r) \oplus (m \parallel 0)) \oplus r) : \text{Guess} \quad [\text{Hash}]}{\vdash_{1/2} f(r'' \parallel H(r') \oplus r \parallel m \vdash_{C_1} r' \parallel f(r'' \parallel r)}
\]

Figure 2.5: Derivations for CPA security of OAEP under one-wayness and partial-domain one-wayness

the event \( \text{Ask}(G, r) \) can be closed by applying rule [\text{Rnd}] because the challenge ciphertext does not depend on \( r \) anymore. For the branch corresponding to the event \( \text{Ask}(H, r') \) we clearly have that

\[
r' \parallel m \vdash_{C_1} r' \\quad f(r'' \parallel r) \parallel m \vdash_{C_1} f(r'' \parallel r).
\]

for \( C_1 = [\oplus]_{0}^{r'} \) and \( C'_1 = [\oplus]_{0}^{r' + |r|} \). Intuitively, querying \( H \) on \( r' \) amounts to inverting \( f \) in its \( \ell = |r'| \) most significant bits, so we conclude applying \([\text{OW}^2(C_1, C'_1)]\).

The derivation for \( \text{OW} \) is depicted at the top of Figure 2.5 and it proceeds by applying rule \([\text{Opt}]\) to move the expression \( H(r') \) from the challenge expression to the event, that becomes \( \text{Ask}(G, H(r') \oplus r) \). We then apply \([\text{Hash}]\) to replace \( H(r') \) by a fresh random bitstring \( r'' \). Note that this rule performs a substitution only on the first branch. The branch corresponding to event \( \text{Ask}(G, r'' \oplus r) \) is closed by first using rule \([\text{Opt}]\) to replace \( r'' \oplus r \) by \( r'' \), so that the event no longer depends on the challenge ciphertext, and then concluding with rule \([\text{Rnd}]\). Finally, to close the last branch, observe that

\[
r' \parallel H(r') \oplus r \parallel m \vdash_{C_2} r' \parallel r \\quad f(r'' \parallel r) \parallel m \vdash_{C_2} f(r'' \parallel r)
\]
Chapter 2. Automated analysis of padding-based encryption

\[ \nabla_{\text{Ask}} \]

\[
\frac{f(r'' \parallel r') : \text{Ask}(G, r'') [\text{OW}_0^0(C, C')]}{\frac{f(r'' \parallel r') : \text{Ask}(H, m \parallel r)}{\frac{f(H(m \parallel r) || r' \oplus (m \parallel r)) : \text{Ask}(G, H(m \parallel r))}{\text{Opt}}} \frac{f(H(m \parallel r) || r'') : \text{Ask}(H, m \parallel r)}{\text{Opt}}} \frac{f(H(m \parallel r) || r' \oplus (m \parallel r)) : \text{Guess}}{\text{Ind}} \frac{f(H(m \parallel r) || r'') : \text{Guess}}{\text{Rand}^f} \frac{f(H(m \parallel r) || r' \oplus (m \parallel r)) : \text{Guess}}{\text{Hash}_1} \]

\[ \nabla_{\text{PSS-E}} \]

\[
\frac{\nabla_{\text{OAEP}}}{\frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{s-pd-\text{OW}}_{\Theta}(0, \ell, q_H, t_A + q_H)}{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)} \frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)}{\frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)}{\frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)}{\text{Opt}}}} \frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)}{\text{Hash}_1}} \frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)}{\text{Opt}} \frac{1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1)}{\text{Hash}_1} \]

Figure 2.6: A derivation for CPA security of PSS-E

for \( C_2 = [\varepsilon_0^{r''} || (H([\varepsilon_0^{r''}] \oplus [\varepsilon_0^{r''}])] \) and \( C'_2 = [\varepsilon_0^{r''} + r'] \). Hence, a CPA adversary that triggers this event can be used to fully invert \( f \), and we can conclude by applying rule \([\text{OW}_0^{t+k}(C_2, C'_2)]\).

Concrete security bound. Applying the bound function \( B_{(t_A, (q_G, q_H))} \) to the derivation \( \nabla_{\text{OAEP}} \), \( \nabla_{\text{PDOW}} \) gives

\[ 1/2 + q_G \times 2^{-k} + \text{Succ}^{s-pd-\text{OW}}_{\Theta}(0, \ell, q_H, t_A + q_H). \]

Applying the bound function \( B_{(t_A, (q_G, q_H))} \) to the derivation \( \nabla_{\text{OAEP}}, \nabla_{\text{OW}} \) gives (after applying the standard reduction from set one-wayness to one-wayness)

\[ 1/2 + q_G \times 2^{-k} + \text{Succ}^{\text{OW}}_{\Theta}(t_A + q_G \times t_1). \]

2.3.5.2 PSS-E

We prove the chosen-plaintext security of PSS-E ([48]) by exhibiting derivations in the CPA. Moreover, we illustrate how to derive concrete security bounds from these derivations. The encryption algorithm of PSS-E is given by the expression

\[ f(H(m \parallel r) || G(H(m \parallel r)) \oplus (m \parallel r)), \]

where \(|m| = k, |r| = \ell, \) and \(|f| = n + k + l, |H(\cdot)| = n, \) and \(|G(\cdot)| = k + l. \)

The derivation for chosen-plaintext security is depicted in Figure 2.6. The proof starts by applying the rule \([\text{Hash}_1] \), replacing the hash \( G(H(m \parallel r)) \) by a fresh random bitstring \( r'' \).
2.4. Chosen-Ciphertext Security

In the branch corresponding to the Guess event, we apply optimistic sampling, replacing \( r' \oplus (m \parallel r) \) by \( r' \), and then \([\text{Hash}_1]\) again, obtaining two premises. The premise corresponding to the original Guess event is proved using rule \([\text{Ind}]\) since the challenge ciphertext no longer depends on \( m \). The premise corresponding to the failure event can be discharged using \([\text{Rnd}]\) because \( m \parallel r \parallel m \vdash* r \) and \( r \) does not appear in the challenge ciphertext, meaning that the adversary would have to guess \( r \) to trigger failure.

The derivation \( \nabla \) of the remaining premise of the first application of rule \([\text{Hash}_1]\) is obtained using a similar strategy. We first apply \([\text{Opt}]\) followed by \([\text{Hash}_1]\), resulting in two premises:

\[- \vdash_0 f(r'' \parallel r) : \text{Ask}(G, r'')\]
\[- \vdash_0 f(r'' \parallel r) : \text{Ask}(H, m \parallel r)\]

To prove the first premise, we use \([\text{OW}]\) since any adversary that asks \( r'' \) to \( G \) can be used to invert \( f \) on its \( n \) most significant bits. The second premise corresponds to the same situation encountered before, where the adversary must query \( m \parallel r \) to \( H \) to trigger failure, but \( r \) does not occur in the challenge ciphertext. We conclude as before using \([\text{Rnd}]\).

**Concrete security bound.** Applying the bound function \( B(t_A, (q_G, q_H)) \) to the derivation \( \nabla_{PSS-E} \) yields

\[
\frac{1}{2} + 2q_H \times 2^l + \text{Succ}_{c^p}^{s-pd-OW}(0, n, q_G, t_A + q_H).
\]

2.4 Chosen-Ciphertext Security

This section introduces a logic for proving chosen-ciphertext security of padding-based encryption schemes by reduction to chosen-plaintext security. This is achieved by systematically transforming the decryption oracle to be able to compute its result from the hash queries performed by the adversary, without using the secret key. At this point, we can completely simulate the decryption oracle and therefore resort to the CPA logic.

2.4.1 Judgments

Judgments are of the form \( \vdash_{\tilde{p}} (c^*, D) : \phi \), where \( \tilde{p} \) is a probability tag, \( c^* \) is an expression, \( D \) is a decryption oracle, and \( \phi \) is an event. The challenge ciphertext \( c^* \) and the tag \( \tilde{p} \) can take the same values as in the CPA logic, whereas the decryption oracle is drawn from the following grammar

\[
F ::= \text{find } x \text{ in } L_H, F \mid \diamond
\]
\[
T ::= e = e' \mid \text{Ask}(H, e) \mid T \land T
\]
\[
D ::= F : T \triangleright e
\]

We assume that \( D \) does not contain random bitstrings and that all variables in expressions in \( D \) are bound by \text{find}, except for a distinguished parameter \( e \) denoting the ciphertext queried to the oracle. The above grammar is sufficiently expressive to encode decryption
Chapter 2. Automated analysis of padding-based encryption

algorithms of padding-based schemes, as well as plaintext-simulators used in reductions. In
formally, given as input a ciphertext \( c \), an algorithm \( \text{find } \vec{x} \in L_H : T \triangleright e \) searches among
the queries made by an adversary to oracles \( H \) for values \( \vec{v} \) satisfying \( T \). If such values
are found, it returns the value of \( e \{ \vec{v}/\vec{x} \} \); otherwise, it returns \( \perp \). Typing is extended
in the obvious way to accomodate decryption oracles. Additionally, the function \( \langle \_ \rangle_C \)
is extended in a straightforward way to evaluate searches on oracle calls: guards are evaluated
as events and finds on queries are interpreted by iterating through the list of adversary
oracle queries.

Events of the logic are as in the CPA logic or of the form \( \exists x \in L_D. T \). For this kind of
event, the quantified variable ranges over the ciphertexts queried to the decryption oracle
during the second phase of the CCA experiment; tests are interpreted as described above.
The remaining events are defined as in the CPA logic.

Judgments are of the form \( \models_{\hat{p}} (e^*, D) : \phi \), where \( e^* \) is a well-typed expression with
\( X(e^*) \subseteq m \), \( D \) is a well-typed decryption oracle, \( \phi \) is an event and \( \hat{p} \) is a probability tag.
For a given trapdoor permutation scheme with permutation generator \( PG \) and adversary
\( A \), we let \( \Pr \llbracket (e^*, D) : \phi \rrbracket_{PG,A} \) denote the success probability of an experiment similar
to the one in Figure 2.2, but where \( A \) has additional access to the decryption oracle \( \langle D \rangle_C \). As in the CPA logic, we define \( \Pr \llbracket (e^*, D) : \phi \rrbracket_{PG,A} \) to be the maximum probability
of \( \Pr \llbracket (e^*, D) : \phi \rrbracket_{PG,t_A,\vec{q}} \) where \( A \) ranges over adversaries executing within time \( t_A \)
and performing at most \( \vec{q} \) queries. The logic ensures that if \( \models_{\hat{p}} (e^*, D) : \phi \) is derivable, then
this probability is upper-bounded by \( \hat{p} \) plus a negligible term (see § 2.4.4).

Extended CPA logic The overall goal of the CCA logic is to show that the decryption
oracle can be replaced by a variant that does not use the secret key, and that the adversary
can distinguish between these with negligible probability. If this is the case, then we can
simulate the decryption oracle and resort to the CPA logic. However, we still need to keep
track of the list of decryption queries the original adversary makes. Therefore, we define
the extended CPA experiment exactly as in Figure 2.2, with the exception that \( A_2 \) returns
a pair \( (\vec{b} , L_D) \). We use the latter to interpret events of the form \( \exists x \in L_D. T \).

2.4.2 Rules

Figure 2.7 presents the rules of the logic. The overall strategy to construct a proof of CCA
security using the logic is to bound the probability of the adversary distinguishing between
the original decryption oracle and a modified oracle that does not use the secret key to
compute its results, but rather reconstructs the plaintext by looking at the queries that
the adversary made to the hash oracles. Once the decryption oracle is public, we resort to
the extended CPA logic to conclude (see § 2.4.5). The rule \( \text{[Reject]} \) allows to transform the
decryption oracle so that it rejects ciphertexts whose decryption requires to make oracle

\[ A \] A plaintext-simulator is an algorithm that extracts the plaintext from a ciphertext by reconstructing
it from oracle queries made by an adversary, without using the trapdoor to the underlying permutation; a
plaintext-simulator that does not query any hash oracle is called a plaintext-extractor.
2.4. Chosen-Ciphertext Security

CCA Judgement Rules:

\[
\begin{align*}
|\text{Pub}(F, T, t)| & \vdash_{\hat{B}} c^*: \phi \quad \text{ST}_{f^{-1}}(T, t) = \emptyset \\
|\text{Conv}| & \vdash_{\hat{B}} (c_1^*, F : T \triangleright t_2) : \phi_2 \\
& \quad \quad (c_1^* = E c_2^* \land \phi_1 \implies E \phi_2 \land \text{ST}_H(c^*) = \{H(e^*)\} \land \text{ST}_H(T, t) = \{H(e)\}) \\
|\text{Find}| & \vdash_{\hat{B}} (c^*, \text{find } x \in \text{L}_H, F : T \{x/e\} \land x = e \triangleright t \{x/e\}) : \phi \\
& \quad \quad \quad \quad \vdash_{\hat{B}} (c^*, F : T \land \text{Ask}(H, e) \triangleright t) : \phi \\
|\text{False}| & \vdash_{\hat{B}} (c^*, D) : \exists c \in \text{L}_D. c = c^* \\
|\text{Reject}| & \vdash_{\hat{B}} (c^*, T \land \text{Ask}(H, e) \triangleright t) : \exists c \in \text{L}_D. T \land e^* = e \\
& \quad \quad \quad \quad \vdash_{\hat{B}} (c^*, T \triangleright t) : \phi
\end{align*}
\]

Test Proof Rules:

\[
\begin{align*}
|\text{PAnd}| & \vdash_{\hat{B}} r \triangleleft : T_i \\
& \quad \quad \quad \quad \vdash_{\hat{B}} r \triangleleft : T_1 \land T_2 \\
|\text{PEqs}(\ell)| & \vdash_{\hat{B}} r \triangleleft : e = e' \\
& \quad \quad \quad \quad \vdash_{\hat{B}} r \triangleleft : e = e' \land r \not\in R(e') \land c \parallel e \vdash [r]_k^\ell \\
|\text{PRnd}(\ell, H)| & \vdash_{\hat{B}} r \triangleleft : \text{Ask}(H, e) \\
& \quad \quad \quad \quad \vdash_{\hat{B}} r \triangleleft : \text{Ask}(H, e) \quad c \parallel e \vdash [r]_{H}^\ell
\end{align*}
\]

Figure 2.7: Proof rules for CCA judgments and proof rules for tests

queries that have not yet been made by the adversary. The resulting decryption oracle no longer makes new queries to the given oracle \(H\). It suffices to consider two events that can lead the adversary to distinguish between the original and transformed oracles: 1. when the adversary makes a decryption query \(c\) such that \(T\) succeeds after making a query \(H(e)\) that is also needed to compute the challenge ciphertext; 2. when the test \(T\) succeeds, even though it gets a random answer from \(H\). The probability of this last event can be proven negligible using the rules [PAnd], [PEqs], and [PRnd] in Figure 2.7.

The rule [False] captures the fact that the adversary cannot ask for the decryption of \(c^*\) during the second phase of the CCA experiment; the probability of this happening is 0. Rule [Conv] allows switching between observationally equivalent decryption oracles and weakening the event considered. This rule is usually used to transform a test that requires \(f^{-1}\) into an equivalent test that only requires \(f\), so that [Pub] becomes applicable.
Chapter 2. Automated analysis of padding-based encryption

The rule [Find] allows replacing an oracle that computes a value explicitly by one that searches for it among oracle queries made by the adversary.

\[
[\text{Eqs}(\vec{r})] \quad \frac{r_i \notin \mathcal{R}(c^*, e) \land \text{dist}(\vec{r})}{\exists c : e \equiv \vec{r} \quad \vec{e} \parallel \mathcal{R}(c^*) \parallel m \vdash \text{dist}(\vec{r})}
\]

\[
[\text{Rnd}(\vec{r})] \quad \frac{\exists c : e \in L_D. \text{Ask}(H_1, e_1) \land \cdots \land \text{Ask}(H_n, e_n)}{r_i \notin \mathcal{R}(c^*) \land \text{dist}(\vec{r})}
\]

\[
[\text{OW}_x^y(C, C')] \quad \frac{\exists c : e \in L_D. \text{Ask}(H_1, e_1) \land \cdots \land \text{Ask}(H_n, e_n)}{f(\vec{r}) \parallel \vec{r}_1 \parallel m \vdash \text{dist}(\vec{r}_1) \land \vec{e}_2 \parallel \vec{r}_2 \parallel m \vdash \text{dist}(\vec{r}_2) \land \vec{e}_2 \parallel \vec{r}_2 \parallel m \vdash \text{dist}(\vec{r}_2) \land \vec{e}_1 \parallel \vec{r}_1 \parallel m \vdash \text{dist}(\vec{r}_1) \land \mathcal{R}(c^*, \vec{e}) = \vec{r}_1 \cup \vec{r}_2}
\]

Figure 2.8: Proof rules of the extended CPA logic

The rule [Pub] links the CCA logic with the CPA logic, and captures the intuition that an adversary does not gain any advantage from getting access to a publicly simulatable decryption oracle. Note that the judgment in the premise may be of the form \( \vdash \phi \), where \( \phi \) is an event of the CCA logic. This kind of judgment is handled by the rule [Eqs] and the generalized rules [Rnd] and [OW] given in Figure 2.8.

2.4.3 Bounds

Security bounds are computed from a derivation \( \triangledown \) in the CCA logic by extending the function \( B \) to handle the new rules. Recall that the function is parametrized by the execution time \( t_A \) and the number of oracle queries \( \vec{q} \) of the adversary. The extended definition is provided in Figure 2.9. Again, we omit resource parameters when they remain unchanged and maintain the previous convention for sub-derivations.

As usual, for all bridging rules, e.g. rule [Find], the bound is inherited from their single premise. For the rule [Reject], the bound is computed as \( p_1 + p_2 + p_3 \), where \( p_1 \) is the probability of the original event in the transformed experiment, \( p_2 \) and \( p_3 \) bound the probability of the two different failure events, one of which can be triggered by any decryption query.

The case of rule [Pub] represents a reduction from CCA to CPA, where a simulator \( S \) uses a CCA adversary \( A \) to win in the CPA experiment. We therefore have to compute bounds for the computational resources \( (t_S, \vec{q}^S) \) used by \( S \) in terms of the resources of \( A \) and the plaintext-simulator \( F : T \triangleright u \). We first define a function \( T \) that computes a bound for the time required to evaluate a test \( T \) or an expression \( u \). Then, \( t_S \) can be defined as \( t_A + q_D \times t_D \) where \( t_D \) is the the time required to evaluate the test \( T \) on each combination of queries traversed by \( F \), plus the time required to evaluate the answer \( u \). Similarly, we
2.4. Chosen-Ciphertext Security

\[
B_{(t_A, \vec{q})}(\nabla) = \begin{cases} 
B(\nabla_1) + B(\nabla_2) + q_D B(\nabla_3) & \text{if } L_\nabla = [\text{Reject}] \\
B_{(t_S, \vec{q}^S)}(\nabla_1) & \text{if } L_\nabla = [\text{Pub}(F, T, u)] \\
0 & \text{if } L_\nabla = [\text{False}] \\
q_H \times 2^{-\ell} & \text{if } L_\nabla = [\text{PRnd}(\ell, H)] \\
2^{-\ell} & \text{if } L_\nabla = [\text{Eqs}(\ell)] \\
B(\nabla_1) & \text{otherwise}
\end{cases}
\]

\[t_S = t_A + q_D \times (T(u) + T(T) \times \prod_{H_j \in F} q_{H_j})\]
\[q_{H_i}^S = q_{H_i} + q_D \times (Q_i(u) + Q_i(T) \times \prod_{H_j \in F} q_{H_j})\]

Figure 2.9: Extended computation of concrete security bounds for CCA derivations.

define a function \(Q_i\) that bounds the number of queries made to \(H_i\) during the simulation of the decryption oracle, and use it to compute \(q_{H_i}^S\).

The \([\text{False}]\) yields exact bounds that can be readily used. The cases of rules \([\text{PRnd}]\), \([\text{Eqs}]\), \([\text{PEqs}]\), correspond to the probability of guessing a random bitstring of length \(\ell\) in \(q_H\), or just 1 tries.

2.4.4 Soundness

Let \(c^*\) be an expression, \(D\) a decryption oracle, \(\phi\) an event and \(\hat{\rho}\) a probability tag.

Theorem 2.3. If \(\nabla\) is a derivation of \(\models_{\hat{\rho}} (c^*, D) : \phi\) under \(\Gamma\), then

\[
\Pr[(c^*, D) : \phi]_{PG, t_A, \vec{q}} \leq B_{(t_A, \vec{q})}(\nabla)
\]

We defer the proof of this lemma to §2.7.

Corollary 2.4. If \(\langle c^* \rangle_C\) is the encryption algorithm and \(\langle D \rangle_C\) is the decryption algorithm of an encryption scheme \(\Pi\); and \(\nabla\) is a derivation of \(\models_{1/2} (c^*, D) : \text{Guess}\) under \(\Gamma\), then

\[
\text{Adv}_{\Pi}^{\text{CCA}}(t_A, \vec{q}) \leq 2 B_{(t_A, \vec{q})}(\nabla) - 1.
\]

Moreover, this bound is negligible if \(t_A\) and all \(q_H\) in \(\vec{q}\) are polynomial in the security parameter, and all assumptions in \(\Gamma\) hold.

Proof sketch. Follows from Theorem 2.3 and the observation that

\[
\text{Adv}_{\Pi}^{\text{CCA}}(t_A, \vec{q}) = 2 \Pr[(c^*, D) : \text{Guess}]_{PG, t_A, \vec{q}} - 1
\]

since \(\langle c^* \rangle_C\) is the encryption algorithm and \(\langle D \rangle_C\) is the decryption algorithm of \(\Pi\).
Chapter 2. Automated analysis of padding-based encryption

\[ \nabla \phi_2 \]

\[
\vdash_0 f(r'' \parallel r'_3' \parallel r'_2') : \phi_5 \quad \text{[Rnd']}
\]

\[
\vdash_0 f(r'' \parallel r') : \phi_4 \quad \text{[Split], [Opt]}
\]

\[
\vdash_0 f(r'' \parallel r') : \phi_4 \quad \text{[Opt], [Conv]}
\]

\[
\vdash_0 f(H(m \parallel r) \parallel r') : \phi_4 \quad \text{[Opt], [Conv]}
\]

\[
\vdash_0 f(H(m \parallel r) \parallel (m \parallel r)) : \phi_3 \quad \text{[Opt], [Conv]}
\]

\[
\vdash_0 (c, D_4) : \phi_2 \quad \text{[Find]}
\]

\[
\vdash_0 (c, D_3) : \phi_2 \quad \text{[Find]}
\]

\[
\vdash_0 c' \vdash \phi_2 \quad \text{[Hash_1]}
\]

\[
\vdash_0 \neg \phi_2 \quad \text{[Hash_1]}
\]

\[
\vdash_0 (c', D_3) : \phi_2 \quad \text{[Find]}
\]

\[
\vdash_0 (c', D_4) : \phi_2 \quad \text{[Hash_1]}
\]

\[
\vdash_0 (c', D_2) : \phi_2 \quad \text{[Hash_1]}
\]

\[
\vdash_0 (c', D_2) : \phi_1 \quad \text{[Conv], [False]}
\]

\[
\vdash_0 r' \vdash \phi_2 \quad \text{[PAnd_2]}
\]

\[
\vdash_0 t = f^{-1}(c) \quad \text{[PRnd(k-k_1, H)]}
\]

\[
\vdash_0 \phi_1 \quad \text{[PEqs(k_1)]}
\]

\[
\vdash_0 t = f^{-1}(c) \quad \text{[Reject]}
\]

\[
\phi_1 \triangleq \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in let } x = G(s) \oplus t \text{ in } s = H(x) \wedge x = m \parallel r
\]

\[
\phi_2 \triangleq \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in let } x = G(s) \oplus t \text{ in } s = H(x) \wedge \text{Ask}(H, x) \wedge s = H(m \parallel r)
\]

\[
\phi_3 \triangleq \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in let } x = r' \oplus t \text{ in } s = H(x) \wedge \text{Ask}(H, x) \wedge s = H(m \parallel r)
\]

\[
\phi_4 \triangleq \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{Ask}(H, r' \oplus (m \parallel r) \oplus t)
\]

\[
\phi_5 \triangleq \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{Ask}(H, (r'_3 \parallel m \parallel [x]_0^n)) \parallel r)
\]

Figure 2.10: A derivation for CCA security of PSS-E

2.4.5 Example

As an example of the use of the IND-CCA logic we present a derivation of CCA security of PSS-E in Figure 2.10. For the sake of clarity, we use let notation and pattern matching to avoid explicit projections and repetition.

Let \( n = k - k_1 - k_0 \). Initially, we have

\[
c^* = f(H(m \parallel r) \parallel G(H(m \parallel r)) \oplus (m \parallel r))
\]

\[
D_1 = \text{let } s \parallel t = f^{-1}(c) \text{ in let } x = G(s) \oplus t \text{ in } s = H(x) \triangleright [x]_0^n
\]

We are interested in proving the judgment \( \vdash_{1/2} (c^*, D_1) : \text{Guess} \). We begin by applying rule \( \text{[Reject]} \) to strengthen the test in the decryption oracle to check also that \( \text{Ask}(H, x) \) holds, so as to reject ciphertexts that would require new queries to \( H \). The corresponding algorithm is

\[
D_2 = \text{let } s \parallel t = f^{-1}(c) \text{ in let } x = G(s) \oplus t \text{ in } s = H(x) \wedge \text{Ask}(H, x) \triangleright [x]_0^n
\]

30
2.4. Chosen-Ciphertext Security

Note that an adversary is able to distinguish between $D_1$ and $D_2$ only if it can produce a ciphertext $c$ that passes the test without querying $H(x)$ in the first place. This is only possible if the adversary either guesses this hash or learns something about it from the challenge ciphertext. The first case corresponds to the premise $\vdash_0 r' \nvdash : \text{let } s \parallel t = f^{-1}(c) \text{ in } s = r'$ and can be proven using $[\text{PEqs}(k_1)]$. The second case corresponds to the premise

\begin{align*}
\vdash_0 (c^*, D_2) : \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{let } x = G(s) \oplus t \text{ in } s = H(x) \land x = m \parallel r
\end{align*}

and is proven using rules $[\text{False}]$ and $[\text{Conv}]$, because the event is equivalent to $\exists c \in L_D. c = c^*$.

To prove the premise corresponding to the original $\textbf{Guess}$ event, we apply rule $[\text{Reject}]$ again, but this time with $G(s)$. We obtain the following decryption oracle

\begin{align*}
D_3 &= \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{let } x = G(s) \oplus t \text{ in } s = H(x) \land \text{Ask}(H, x) \land \text{Ask}(G, s) \triangleright [x]_0^n
\end{align*}

The premise

\begin{align*}
\vdash_0 r' \nvdash : \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{let } x = r' \oplus t \text{ in } s = H(x) \land \text{Ask}(H, x)
\end{align*}

can be readily discharged using the rules $[\text{PAnd}_2]$ and $[\text{PRnd}(k - k_1, H)]$.

To prove the two premises corresponding to $\textbf{CCA}$ judgments, we first apply rules $[\text{Find}]$ and $[\text{Conv}]$ to reformulate the decryption oracle as follows:

\begin{align*}
D_4 &= \text{find } s, x \text{ in } L_G, L_H : c = f(s \parallel G(s) \oplus x) \land s = H(x) \triangleright [x]_0^n
\end{align*}

Note that this decryption oracle is $\textit{public}$, i.e. it does not use $f^{-1}$ and can be efficiently simulated. Hence, we can apply rule $[\text{Pub}]$ and proceed reasoning in the (extended) CPA logic. The branch corresponding to the original $\textbf{Guess}$ event can be proven using the same derivation $\nabla_{\text{PSS-E}}$ in Figure 2.6 that we used to prove CPA security.

We have only one outstanding goal,

\begin{align*}
\vdash_0 c^* : \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{let } x = G(s) \oplus t \text{ in } s = H(x) \land \text{Ask}(H, x) \land s = H(m \parallel r)
\end{align*}

To prove it, we first apply $[\text{Hash}_1]$ to replace $G(H(m \parallel r))$ with a fresh random bitstring $r'$. The derivation $\nabla_{\text{Ask}}$ that bounds the probability of failure is as in the proof of CPA security in Figure 2.6. We apply rules $[\text{Opt}]$ and $[\text{Conv}]$ to simplify the premise corresponding to the original event, obtaining

\begin{align*}
\vdash_0 f(H(m \parallel r) \parallel r') : \exists c \in L_D. \text{let } s \parallel t = f^{-1}(c) \text{ in } \text{Ask}(H, r' \oplus (m \parallel r) \oplus t)
\end{align*}

We then apply $[\text{Hash}_1]$ again, this time to replace $H(m \parallel r)$ with a fresh random bitstring $r''$. The premise corresponding to the failure event can be readily proved using $[\text{Rnd}]$ as in the proof of CPA security. To prove the remaining premise, observe that to trigger the event $\text{Ask}(H, r' \oplus (m \parallel r) \oplus t)$, the adversary should be able to compute $[r' \oplus t]_n \oplus r$, but $r$ is now independent of the challenge ciphertext. We formalize this reasoning by first splitting $r'$ and then applying rule $[\text{Opt}]$ to replace $[r' \oplus t]_n \oplus r$ by simply $r$. We conclude applying the generalized $[\text{Rnd}]$ rule, with $C_4 = [\triangleright]^k$. 

31
Chapter 2. Automated analysis of padding-based encryption

Concrete security bound. The concrete security bound \( B_{(t_A, (q_G, q_H, q_D))}(\nabla_{\text{PSS-E}}) \) obtained from this proof is

\[
B_{(t_S, (q_G, q_H, q_D))}(\nabla_{\text{PSS-E}}) + B_{(t_S, (q_G, q_H, q_D))}(\nabla_{\phi_2}) + q_D \times (2^{-k_1} + q_H \times 2^{k_1-k})
\]

where \( t_S = t_A + q_D q_G q_H \times t_t \), and

\[
B_{(t_S, (q_G, q_H, q_D))}(\nabla_{\text{PSS-E}}) \approx 1/2 + 2q_H \times 2^{-k_0} + \text{Succ}_{\Theta}^{s-pd-OW}(k_1, q_G, t_A + q_D q_G q_H \times t_t)
\]

\[
B_{(t_S, (q_G, q_H, q_D))}(\nabla_{\phi_2}) = 3q_H \times 2^{-k_0} + \text{Succ}_{\Theta}^{s-pd-OW}(k_1, q_G, t_A + q_D q_G q_H \times t_t)
\]

Observe that \( t_S \) comes from the two applications of rule [Find] and can be improved to \( t_A + q_D q_H \times t_t \), because to answer a decryption query, the simulator \( D_4 \) just needs to traverse \( L_H \) to find a value for \( x \), the value of \( s \) is determined by \( H(x) \). The overall bound is, ignoring constant factors,

\[
1/2 + q_H \times 2^{-k_0} + q_D \times 2^{-k_1} + q_D \times q_H \times 2^{k_1-k} + \text{Succ}_{\Theta}^{s-pd-OW}(k_1, q_G, t_A + q_D q_H \times t_t)
\]

In comparison, the bound given by Coron [48] is:

\[
1/2 + q_H \times 2^{-k_0} + q_D \times 2^{-k_1} + \text{Succ}_{\Theta}^{s-pd-OW}(k_1, q_G + q_H, t_A + q_D q_H \times t_t)
\]

The differences are due to the use of a slightly different reduction, where the hash oracle \( H \) is assumed to issue a query \( G(s) \) each time it produces a response \( s \). This assumption would make redundant the application of rule [Reject] in \( \nabla_{\text{Guess}} \) in our reduction, eliminating the terms coming from \( \nabla_{\phi_2} \) and \( q_D \times q_H \times 2^{k_1-k} \). The resulting bound, once adjusting \( q_G \) for the additional queries issued by \( H \), coincides with that of Coron et al.

2.5 Attacks

This section describes our approach for finding attacks against chosen-plaintext and chosen-ciphertext security of padding-based encryption schemes. Since our logics are incomplete, we use attack finding to obtain negative results, and additional data points to evaluate schemes for which we cannot obtain proofs.

We distinguish between universal attacks and existential attacks relative to a set of assumptions. An attack is universal if it works against every possible instantiation of the trapdoor permutations used by a scheme. An attack is existential if it works for some trapdoor permutation consistent with the assumptions, i.e. it may rely on specific properties of the employed trapdoor permutation. We rely on Static Distinguishability to find attacks.

2.5.1 Inequality and Static Distinguishability

Static equivalence [1] models symbolic indistinguishability of expressions. Informally, two expressions \( e_0 \) and \( e_1 \) are statically equivalent if all tests of the form \( C\{\cdot/\square\} =_E C'\{\cdot/\square\} \)
s ⊭ t ∈ \Gamma \quad \frac{C[s] \neq C[t]}{s \neq t} \quad \frac{s \neq u \quad t = E \quad u}{s \neq t} \quad \text{[Ax]} \quad \frac{s \neq t}{s \neq t} \quad \frac{C \in \mathcal{C}}{s \neq t} \quad \frac{s \neq u \quad t = E \quad u}{s \neq t} \quad \text{[EqE]}

r \notin \mathcal{R}(s) \quad \frac{e \neq e_1 \quad \cdots \quad e \neq e_k}{H(e) \neq H(s)} \quad \text{[Rnd]} \quad \frac{H(e) \neq \bigoplus_{i=1}^{k} H(e_i)}{s \neq t} \quad \frac{\oplus_{i=1}^{k} H(e_i)}{s \neq t} \quad \text{[Hash]}

Figure 2.11: Proof rules for apartness \neq

succeed for \( e_0 \) iff they succeed for \( e_1 \). We use a relation inspired by static equivalence to model distinguishability of expressions. We call this relation static distinguishability, and define it in terms of an apartness relation \( \neq \) in place of \( \neq E \). The soundness of this relation implies that if \( s \neq t \) holds, then the probability that the results of \( L_s M \) and \( L_t M \) coincide when shared random bitstrings are jointly sampled is negligible. For a given set of axioms \( \Gamma \), the relation \( \neq \) is inductively defined by the rules given in Figure 2.11.

We use axioms of the form

\[ \Gamma \subseteq \{ s \neq t \mid s, t \in \{0\} \cup \mathcal{X} \} \]

to model assumptions about inequalities of variables. For example, \( \Gamma \) can formalize that all considered variables are pairwise distinct and distinct from 0.

**Definition 2.5** (Static Distinguishability). The relation \( \neq \) is the smallest symmetric relation on well-typed expressions such that \( e_0 \neq e_1 \) iff there are contexts \( C \) and \( C' \) such that

i) \( \mathcal{R}(C, C') = \emptyset \), \( \mathcal{A}(C) \subseteq \{\square\} \); 

ii) \( f^{-1} \) does not occur in \( C, C' \), and

iii) \( C \{ e_0 / \square \} \neq E \quad C' \{ e_0 / \square \} \) and \( C \{ e_1 / \square \} \neq C' \{ e_1 / \square \} \).

We use static distinguishability to find attacks on encryption schemes. For the attacks to be meaningful, we rely on the soundness of \( \neq \), which follows from the soundness of \( =E \) and \( \neq \). Informally, if \( e_0 \neq e_1 \) holds, then there is an efficient algorithm distinguish that when given as input \( f \) and a value computed using \( \langle e_0 \rangle \) returns 0, but when given instead a value computed using \( \langle e_1 \rangle \) returns 1, except with negligible probability.

As an example, the expressions \( e_0 = f(r) \parallel (r \oplus 0) \) and \( e_1 = f(r) \parallel (r_1 \oplus r_2) \) are statically distinguishable, because for the contexts

\[ C = [\square]^{f(r)} \parallel 0 \quad \text{and} \quad C' = f \left( [\square]^{f(r)} \parallel 1 \right) \]

it holds that \( C \{ e_0 / \square \} = E \quad C' \{ e_0 / \square \} \) and \( C \{ e_1 / \square \} \neq C' \{ e_1 / \square \} \). A simple distinguisher algorithm applies the algorithms corresponding to \( C \) and \( C' \) to its input and returns 0 if this yields equal bitstrings and 1 otherwise.
2.5.2 Universal Attacks

In order to find universal attacks against the CPA security of an encryption scheme with encryption algorithm $e$, we search for closed expressions $m_0$ and $m_1$ that do not contain symbols of the form $f^{-1}$ such that $e\{m_0/m\} \not\approx e\{m_1/m\}$. By soundness of $\not\approx$, there exists an efficient algorithm distinguish that returns 0 for input $\langle e\{m_0/m\} \rangle$ and 1 for input $\langle e\{m_1/m\} \rangle$ with overwhelming probability. To mount an attack against CPA security using this algorithm, an adversary chooses plaintexts $\langle m_0 \rangle$ and $\langle m_1 \rangle$, receives the challenge ciphertext $c^*$, and returns distinguish$(c^*)$. An example of a scheme vulnerable to this attack is a scheme with encryption algorithm given by

$$e = f(r) \parallel (G(r) \oplus m) \parallel H(m)$$

A CPA adversary can use the distinguisher obtained from the contexts $C = [\circ]^{|H(m)|}$ and $C' = H(0^{|m|})$ to tell apart $e\{0/m\}$ from $e\{1/m\}$.

We search for malleability attacks using deducibility to refute CPA security. Specifically, we search for closed expressions $m_0$ and $\Delta \neq 0^{|m|}$ such that $e\{m_0/m\} \vdash_C e\{m_0 \oplus \Delta/m\}$. This would imply an effective attack against CPA security: choose plaintexts $m_0$ and $m_1 \neq m_0$ and obtain the challenge $c^*$; then query the decryption oracle on $\langle C \{c^*/\circ\} \rangle$, xor the result with $\Delta$, and return 0 if it equals $\langle m_0 \rangle$ and 1 otherwise. An example of a scheme vulnerable to a universal attack of this form is the Zheng-Seberry cryptosystem [100], whose encryption algorithm is given by $e = f(r)\parallel (G(r) \oplus (m \parallel H(m)))$; take $m_0 = 0^{|m|}$ and $\Delta = 1^{|m|}$.

2.5.3 Existential Attacks

In order to find existential attacks against a scheme with encryption algorithm $e$ w.r.t. a set of assumptions $\Gamma$, we find universal attacks against modified versions of it. An example can elucidate better the point.

Consider the encryption algorithm of ZAEP [22], $e = f(r) \parallel G(r) \oplus m)$. To show that there is no blackbox reduction from the CPA security of ZAEP to the assumption $(f,0)|f|$, we instantiate $f$ as follows

$$f(a \parallel b) = a \parallel f_2(b) \quad (2.1)$$

If $f_2$ is a one-way permutation, the permutation $f$ satisfies the assumption $(f,0)|f|$. Using static distinguishability, we find an attack on $e' = r \parallel f_2(G(r) \oplus m)$ given by the contexts

$$C = [\circ]^{|m|} \text{ and } C' = f_2\left(G\left([\circ]^{|r|}\right)\right)$$

which can be used to distinguish $e\{0/m\}$ and $e\{1/m\}$.

We can show that there is no blackbox reduction of the CCA security of ZAEP to an arbitrary $\Gamma$, using the following instantiation

$$f(a) = f'(\left[a\right]^{|a|-c} \parallel \left[a\right]^{|c|-c} \quad (2.2)$$
where \( c \) is a size variable that we interpret as a constant. It is easy to see that ciphertexts computed using this instantiation are malleable. Moreover, assuming \( f' \) satisfies the assumptions \( \Gamma \) (accounting for the size reduction by \( c \)), this instance of \( f \) satisfies \( \Gamma \). This is because size variables in assumptions grow polynomially with the security parameter, and leaking a constant fraction of a pre-image cannot be used to attack any assumption in \( \Gamma \).

In contrast, the ability to compute a ciphertext of a message that differs in just one bit from the decryption of a given ciphertext results in a chosen-ciphertext attack.

In general, we can prove that there is no blackbox reduction for a fixed set of one-wayness assumptions \( \Gamma \) by either finding a universal attack, or instantiations for the trapdoor permutations that yield attacks and are compatible with all assumptions in \( \Gamma \). For example, the instantiation (2.2) above is compatible with any set of assumptions, while (2.1) is compatible with all assumptions except those of the form \((f, k, \ell)\) with \( 0 < \ell \) and \( k + \ell \leq |a| \). In addition to the aforementioned instantiations, we also use instantiations of the form

\[
f(a \Vert b \Vert c) = f_1(a) \Vert b \oplus a \Vert c
\]

which allow us to find attacks if \( f \) is used in such a way that part of its input is leaked or interdependent.

### 2.6 Experimental Validation

We implemented the proof search and attack finding methods described in § 2.3, § 2.4 and § 2.5 in a toolset that we coin ZooCrypt. ZooCrypt can prove the CPA and CCA security of a scheme under different assumptions on the trapdoor permutations it uses, or find attacks that are consistent with these assumptions.

#### 2.6.1 Security Analysis

To analyze the security of a scheme with encryption algorithm given by an expression \( c^* \) under a set of assumptions \( \Gamma \), the toolset follows the workflow depicted in Figure 2.12:

i) Checks that \( c^* \) is well-typed and that encryption is invertible, i.e. \( c^* \vdash^* m \);

ii) Searches for attacks against CPA security;

iii) Searches for proofs of CPA security. If a proof is found, computes the corresponding security bound;

iv) Searches for malleability attacks against CCA security;

v) If a CPA proof has been found, synthesizes a decryption algorithm \( D \) and searches for a proof of CCA security. If a proof is found, computes the corresponding security bound.

The results of this security analysis are an adversary for each attack found, and derivations for all security proofs together with the set of assumptions effectively used and the corresponding concrete security bound. Steps 3 and 5 implement proof search algorithms.
Chapter 2. Automated analysis of padding-based encryption

Figure 2.12: Security analysis workflow

for the logics. These algorithms try to build a valid derivation bottom up, by applying rules in a prescribed order. Simple heuristics allow to improve the efficiency of the search and to ensure termination.

A crucial step in the above workflow is synthesizing a decryption algorithm that reject as many invalid ciphertexts as possible, with the aim of easing the construction of a plaintext simulator during a CCA analysis. Indeed, an encryption algorithm typically admits several correct decryption algorithms, because the consistency condition gives complete freedom of choice as to what should be the result of decrypting an invalid ciphertext. The tool infers such algorithms using a method inspired by [24, 46] to analyze the redundancy built into ciphertexts. We exploit the fact that our algorithm for checking static equivalence computes as a sub-routine non-trivial equations that hold for an expression; when applied to an expression denoting an encryption algorithm, this sub-routine yields tests for checking the validity of ciphertexts.
2.6. Experimental Validation

2.6.2 Generation of Encryption Schemes

Our tool also implements an algorithm that generates expressions denoting encryption algorithms within budget constraints specified as the number of concatenation, exclusive-or, hash and trapdoor permutation constructors.

Candidate encryption schemes are generated following a top-down approach that uses variables to represent holes in partially specified expressions. Starting from a fully unspecified expression, i.e. just a variable $x$, at each iterative step the tool picks a hole and replaces it with either:

- An expression of the form $f(x)$, $H(x)$, $x \oplus y$ or $x \| y$, for fresh variables $x$ and $y$, if the budget permits;

- A hole-free sub-expression of $e$ or one of $0$, $m$, $r$; this does not consume the budget.

An incremental type-checker is used at each step to discard partially specified expressions that do not have any well-typed instance. For example, $e \oplus (e \| x)$ is immediately discarded because $e$ cannot be assigned a size regardless of any substitution for the hole $x$.

We trim large parts of the search space by implementing an early pruning strategy in the style of [89]. Concretely, we apply some simple filters during generation. For instance, given an expression $e$ with holes, we check for the existence of a substitution $\sigma$ for holes respecting the budget constraints such that $e \sigma \vdash^* m$, and that it is not the case that for all such substitutions $e \sigma \vdash m$ or $e \sigma \| m \vdash \mathcal{R}(e)$. These filters can be implemented efficiently using memoization to drastically reduce their computational cost.

2.6.3 Experiments

We evaluate our tool taking into account two different metrics, that complement each other: scalability and precision.

Scalability tries to assess the performance of the tool by running it on schemes generated randomly. Precision tries to estimate how well our tool performs on schemes where the security results are well known. This provides a measure of how complete the tool is in practical terms and further validates the scalability experiments by reassuring that these results are meaningful.

Scalability. We evaluate our tools on encryption schemes generated under different budget constraints. Figures 2.1 and 2.2 summarize the results of the automated security analysis of §2.6.1. In the figures, schemes are classified in rows by their size, measured by the total number of operators used in the expression denoting their encryption algorithm.

The reported experiment has been conducted under two classes of assumptions:

i) $\Gamma_1 = \{(f, 0, |f|) \mid f \in \mathcal{F}(c^*)\}$, i.e. assuming that all trapdoor permutations are one-way;

ii) $\Gamma_2 = \{(f, k_f, n_f) \mid f \in \mathcal{F}(c^*)\}$ such that $0 \leq k_f$ and $k_f + n_f \leq |f|$ for all $f \in \mathcal{F}(c^*)$, i.e. one arbitrary one-wayness assumption for each trapdoor permutation;
Chapter 2. Automated analysis of padding-based encryption

<table>
<thead>
<tr>
<th>Size</th>
<th>Total</th>
<th>OW CPA (%) of Total</th>
<th>CPA (%) of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(50.00%)</td>
<td>(50.00%)</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(18.18%)</td>
<td>(81.82%)</td>
</tr>
<tr>
<td>6</td>
<td>335</td>
<td>65</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(19.40%)</td>
<td>(80.60%)</td>
</tr>
<tr>
<td>7</td>
<td>3263</td>
<td>510</td>
<td>2735</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(15.63%)</td>
<td>(83.82%)</td>
</tr>
<tr>
<td>8</td>
<td>32671</td>
<td>4430</td>
<td>27894</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(13.56%)</td>
<td>(85.38%)</td>
</tr>
<tr>
<td>9</td>
<td>350111</td>
<td>43556</td>
<td>301679</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(12.44%)</td>
<td>(86.17%)</td>
</tr>
<tr>
<td>10</td>
<td>644563</td>
<td>67863</td>
<td>569314</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(10.53%)</td>
<td>(88.33%)</td>
</tr>
<tr>
<td>Total</td>
<td>1030989</td>
<td>116433</td>
<td>901929</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(11.29%)</td>
<td>(87.48%)</td>
</tr>
</tbody>
</table>

Table 2.1: Evaluation of the tool on generated encryption schemes, CPA security

The columns grouped under OW CPA report the results obtained when analyzing CPA security under $\Gamma_1$. Column Proof indicates the number of schemes proved secure, column Attack the number of schemes for which some attack (existential or universal) was found, and column Undec. the number of schemes for which security could not be decided. Similarly, the columns grouped under CPA and CCA report the results when analyzing CPA and CCA security under all assumptions of the form $\Gamma_2$. In this case, column Proof indicates the number of schemes proved secure under some assumption of the form $\Gamma_2$, column Attack the number of schemes for which an attack was found for all assumptions of the form $\Gamma_2$, and column Undec. the number of schemes for which security could not be decided. The attack and proof search algorithms are extremely efficient, e.g. proof search for schemes of size 7 takes on average 0.1ms for CPA and 0.5ms for CCA on a modern workstation.

Observe that the figures in the first two groups suggest the separation between one-wayness and partial-domain one-wayness: the stronger the assumption, the more schemes can be proven secure and the less attacks can be found.

Finally, column NR in the CCA group counts the number of schemes that are CPA secure but non-redundant, meaning that all ciphertexts are valid. Non-redundant schemes can be CCA secure [19, 69], but their proofs require random oracle programmability, which is out of the scope of our logics.
2.6. Experimental Validation

<table>
<thead>
<tr>
<th>Size</th>
<th>Total</th>
<th>CCA (% of CPA Proof + CPA Undec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Proof</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.00%)</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.00%)</td>
</tr>
<tr>
<td>6</td>
<td>335</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.98%)</td>
</tr>
<tr>
<td>7</td>
<td>3263</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5.05%)</td>
</tr>
<tr>
<td>8</td>
<td>32671</td>
<td>536</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(6.26%)</td>
</tr>
<tr>
<td>9</td>
<td>350111</td>
<td>7279</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(8.16%)</td>
</tr>
<tr>
<td>10</td>
<td>644563</td>
<td>20140</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(11.29%)</td>
</tr>
<tr>
<td>Total</td>
<td>1030989</td>
<td>28001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(10.11%)</td>
</tr>
</tbody>
</table>

Table 2.2: Evaluation of the tool on generated encryption schemes, CCA security

**Precision**  We also evaluated our automated proof search and attack finding algorithms on a number of schemes from the literature, including the over one hundred variants of OAEP and SAEP surveyed by Komano et al. [69]. In all cases, our results are consistent with published results, and in most we are able to prove security under exactly the same assumptions and obtain the same security bounds. As evidence of the effectiveness of our methodology, we observe that our analyses decide the CPA security of all 72 variants of OAEP in the taxonomy of [69]. The results for CCA security are more nuanced: for about 20% of schemes, we fail to find proofs or attacks when they exist. Table 2.3 summarizes our results for the proposed variants of OAEP.

For CPA security our methods seem to achieve empirical completeness, which suggests that completeness may be provable for some class of schemes or for some mild extension. A closer examination of the schemes on which our CCA analysis is unable to decide security reveals that this is either due to the fact that our approximation of inequality in rules $[\text{Eqs}]$ and $[\text{PEqs}]$ is too coarse, or because the schemes are non-redundant.

Non-redundancy complicates enormously the task of simulating the decryption oracle in CCA reductions, because a meaningful response must be returned in all cases. Proving CCA security of non-redundant schemes, requires *programming* random oracles in order to maintain consistency during simulation, something that we have intentionally avoided in our proof systems. Extending our proof systems to embody some form of *programmability* would reduce the number of schemes for which security cannot be decided, and would be a step towards closing the empirical completeness gap.
## Chapter 2. Automated analysis of padding-based encryption

<table>
<thead>
<tr>
<th>Name</th>
<th>OW CPA</th>
<th>CPA</th>
<th>CCA</th>
<th>Name</th>
<th>OW CPA</th>
<th>CPA</th>
<th>CCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAEP/OAEP×-0A</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-0A</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-0B</td>
<td>✓</td>
<td>✓</td>
<td>No-Red</td>
<td>OAEP/OAEP×-0B</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-0C</td>
<td>✓</td>
<td>✓</td>
<td>No-Red</td>
<td>OAEP/OAEP×-0C</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-0D</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-0D</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-0E</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-0E</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-0F</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-0F</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-1A</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-1A</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/OAEP×-1B</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-1B</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-1C</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-1C</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-1D</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-1D</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-1E</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-1E</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-1F</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>OAEP/OAEP×-1F</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-2A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-2A</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-2B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-2B</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-2C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-2C</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-2D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-2D</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-2E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-2E</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-2F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-2F</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP×-3A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-3A</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/OAEP×-3B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-3B</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/OAEP×-3C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-3C</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/OAEP×-3D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-3D</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/OAEP×-3E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-3E</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/OAEP×-3F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/OAEP×-3F</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-0A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-0A</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-0B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-0B</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-0C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-0C</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-0D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-0D</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-0E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-0E</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-0F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-0F</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-1A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-1A</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-1B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-1B</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-1C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-1C</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-1D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-1D</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-1E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-1E</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
<tr>
<td>OAEP/xOAEP-1F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OAEP/xOAEP-1F</td>
<td>✓</td>
<td>✓</td>
<td>(msb)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>OW CPA</th>
<th>CPA</th>
<th>CCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAEP/OAEP-0A</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-0B</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-0C</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-0D</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-0E</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-0F</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-1A</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-1B</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-1C</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-1D</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-1E</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-1F</td>
<td>✓</td>
<td>✓</td>
<td>(ow)</td>
</tr>
<tr>
<td>OAEP/OAEP-2A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-2B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-2C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-2D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-2E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-2F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-3A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-3B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-3C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-3D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-3E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/OAEP-3F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-0A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-0B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-0C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-0D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-0E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-0F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-1A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-1B</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-1C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-1D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-1E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OAEP/xOAEP-1F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.3: Results on variants of OAEP

### 2.7 Proofs

#### 2.7.1 Soundness of CPA logic

**Theorem 2.1.** Let $c^*$ be an expression, $\phi$ an event and $\hat{p}$ a probability tag. If $\nabla$ is a derivation of $\vdash \hat{p} c^* : \phi$ under $\Gamma$, then

$$B_{(t,A,q)}(\nabla) \geq \begin{cases} 2 \Pr[c^*: \phi]_{PG,t,A,q} - 1 & \text{if } \phi = \text{Guess} \\ \Pr[c^*: \phi]_{PG,t,A,q} & \text{otherwise} \end{cases}.$$
Proof. The proof follows by structural induction in the derivation of $\vdash^r c^* : \phi$.

Bridging Rules: Bridging rules are of the form

$$\frac{\vdash^r c_2^* : \phi_2}{\vdash^r c_1^* : \phi_1}$$

and it suffices to show that for all permutation generators $PG$ and adversaries $A$, we know that

$$\Pr[[c_1^* : \phi_1]_{PG,A}] = \Pr[\text{CPA}_{PG,A}(c_1^*, \phi_1) = \text{true}] \leq \Pr[\text{CPA}_{PG,A}(c_2^*, \phi_2) = \text{true}] = \Pr[[c_2^* : \phi_2]_{PG,A}].$$

If this is the case, the left-hand-side can be bounded using the induction hypothesis. For the rules $[\text{Opt}]$, $[\text{Split}]$, and $[\text{Perm}]$, we actually prove equality of the probabilities.

Case $[\text{Opt}]$. We show that the view of the adversary and the distribution of the bit $b$ are equal in the games $\text{CPA}_{PG,A}(c_1^*, \phi_1)$ and $\text{CPA}_{PG,A}(c_2^*, \phi_2)$ for $c_2^* = c_1^* \{r \oplus e/r\}$ and $\phi_2 = \phi_1 \{r \oplus e/r\}$ with $r \notin R(e)$ using an intermediate game $G$. We obtain $G$ from $\text{CPA}_{PG,A}(c_1^*, \phi_1)$ by inserting the assignment $r \leftarrow r \oplus (e|c)$ immediately after sampling $r$. Since $e$ does not contain $r$, the function $r \mapsto r \oplus (e|c)$ is bijective and it follows that the view of the adversary and the distribution of $b$ in $G$ and $\text{CPA}_{PG,A}(c_1^*, \phi_1)$ coincide. To see that the same holds for $G$ and $\text{CPA}_{PG,A}(c_2^*, \phi_2)$, note that $\text{CPA}_{PG,A}(c_2^*, \phi_2)$ can be obtained from $G$ by inlining the assignment $r \leftarrow r \oplus (e|c)$ in $L6$ and $L7$ since $c_2^* = c_1^* \{r \oplus e/r\}$ and $\phi_2 = \phi_1 \{r \oplus e/r\}$.

Case $[\text{Split}]$. We have that $c_2^* = c_1^* \{r_1 \| r_2/r\}$ and $\phi_2 = \phi_1 \{r_1 \| r_2/r\}$. The proof is similar to the previous case and uses an intermediate game $G$. We obtain $G$ from $\text{CPA}_{PG,A}(c_1^*, \phi_1)$ by replacing the sampling of $r$ with the following line:

$$r_1 \leftarrow \{0,1\}^{|r_1|}; r_2 \leftarrow \{0,1\}^{|r_2|}; r \leftarrow r_1 \| r_2;$$

Since $r$ is a uniformly sampled bitstring in both games, the view of the adversary and the distribution of $b$ in $G$ and $\text{CPA}_{PG,A}(c_1^*, \phi_1)$ coincide. To see that the same holds for $G$ and $\text{CPA}_{PG,A}(c_2^*, \phi_2)$, note that $\text{CPA}_{PG,A}(c_2^*, \phi_2)$ can be obtained from $G$ by inlining the assignment $r \leftarrow r_1 \| r_2$.

Case $[\text{Perm}]$. First note that we have $c_1^* = c_2^* \{f(\vec{r})/\vec{r}\}$ and $\phi_1 = \phi_2 \{f(\vec{r})/\vec{r}\}$ for a vector $\vec{r} = r_1 \| \ldots \| r_n$ of distinct random variables that are only used in the context $\vec{r}$ in $c_2^*$ and $\phi_2$. Our proof uses the intermediate game $G$ obtained from $\text{CPA}_{PG,A}(c_2^*, \phi_2)$ by inserting the assignment $r_1 \| \ldots \| r_l \leftarrow f(r_1 \| \ldots \| r_l)$ after line $L5$. Since $f$ is a bijection and $\vec{r}$ is a uniformly sampled bitstring, this does not change the distribution of $\vec{r}$ and $G$ is therefore equivalent to $\text{CPA}_{PG,A}(c_2^*, \phi_2)$. To complete the proof, note that $\text{CPA}_{PG,A}(c_1^*, \phi_1)$ can be obtained from $G$ by inlining the assignment $r_1 \| \ldots \| r_l \leftarrow f(r_1 \| \ldots \| r_l)$. 

41
Chapter 2. Automated analysis of padding-based encryption

Case [Conv]. We define two intermediate games $G_1$ and $G_2$. For $i \in \{1, 2\}$, the games $G_i$ are obtained from $\text{CPA}_{PG, A}(c_i^*, \phi_i)$ by sampling the random variables $R(c_1, \phi_1) \cup R(c_2, \phi_2)$ in line L5. The two games only differ in line L6 where $G_i$ uses $(c_i^*)$ and in line L7 where $G_i$ uses $(\phi_i)$. Soundness of $= \text{E}$ implies that $G_1$ and $G_2$ are equivalent up to line L7 since $c_1^* = \text{E} c_2^*$. Furthermore, we have $\phi_1 \implies \text{E} \phi_2$ which implies that $G_2$ returns true if $G_1$ returns true and hence

$$\Pr[\text{CPA}_{PG, A}(c_1^*, \phi_1) = \text{true}] = \Pr[G_1 = \text{true}] \leq \Pr[G_2 = \text{true}] = \Pr[\text{CPA}_{PG, A}(c_2^*, \phi_2) = \text{true}] .$$

Hash Rules: Hash Rules are of the form

$$\text{Hash Rules:} \quad \frac{\vdash \phi_2}{\vdash \phi_3}$$

and it suffices to show that for all permutation generators $PG$ and adversaries $A$, it holds that

$$\Pr[\text{CPA}_{PG, A}(c_1^*, \phi_1) = \text{true}] \leq \Pr[\text{CPA}_{PG, A}(c_2^*, \phi_2) = \text{true}] + \Pr[\text{CPA}_{PG, A}(c_3^*, \phi_3) = \text{true}] .$$

Case [Hash1]. We have $c_1^* = c_2^* \{H(e)/r\}$, $c_2^* = c_3^*$, $\phi_1 = \phi_2 \{H(e)/r\}$, and $\phi_3 = \text{Ask}(H, e)$. Furthermore, it holds that $c_2^*$, $\phi_2$, and $e$ do not contain applications of $H$ and $r \notin R(e)$. We use the intermediate game $G$ obtained from $\text{CPA}_{PG, A}(c_2^*, \phi_2)$ by replacing the sampling of $r$ by the assignment $r \leftarrow H(\{e\}_C)$ and sampling the missing random variables required by $(c)_C$.

The proof then follows from the following (in)equalities:

$$\Pr[\text{CPA}_{PG, A}(c_1^*, \phi_1) = \text{true}] = \Pr[G = \text{true}] \\ \leq \Pr[\text{CPA}_{PG, A}(c_2^*, \phi_2) = \text{true}] + \Pr[\text{CPA}_{PG, A}(c_2^*, \phi_2) : (c)_C \in L_H] \\ \leq \Pr[\text{CPA}_{PG, A}(c_2^*, \phi_2) = \text{true}] + \Pr[\text{CPA}_{PG, A}(c_2^*, \text{Ask}(H, e)) = \text{true}]$$

The first step follows from $c_1^* = c_2^* \{H(e)/r\}$ and $\phi_1 = \phi_2 \{H(e)/r\}$. The second step uses Shoup’s Fundamental Lemma [95] and exploits that $G$ and $\text{CPA}_{PG, A}(c_2^*, \phi_2)$ are equivalent unless the event $(c)_C \in L_H$ happens, i.e., they are equivalent until the adversary queries $(c)_C$ to $H$. To see why this is the case, first note that the two games only differ in the definition of $r$, the game $G$ uses $r \leftarrow H(\{e\}_C)$ and the game $\text{CPA}_{PG, A}(c_2^*, \phi_2)$ uses $r \notin \{0, 1\}^{|r|}$. If the adversary does not query $(c)_C$ to $H$ in $G$, then $r$ is uniform independent since $H$ is a random function and $H$’s image for $(c)_C$ is only used to define $r$. The third step follows from the semantics of the event $\text{Ask}(H, e)$.

Case [Hash2]. We have $c_1^* = c_2^* = c_3^* \{H(e)/r\}$, $\phi_1 = \phi_2 \{H(e)/r\}$, and $\phi_3 = \phi_2 \{H(e)/r\} \land \text{Ask}(H, e)$. Furthermore, it holds that $c_2^*$, $\phi_2$, and $e$ do not contain applications of $H$ and $r \notin R(e)$. We use the intermediate game $G$ obtained from $\text{CPA}_{PG, A}(c_2^*, \phi_2)$ by replacing the
2.7. Proofs

sampling of \( r \) by the assignment \( r \leftarrow H(\langle e \rangle_C) \) and sampling the missing random variables required by \( \langle e \rangle_C \). The proof then follows from the following (in)equalities:

\[
\Pr[\text{CPA}_{PG,A}(c_1, \phi_1) = \text{true}] = \Pr[G = \text{true}]
\]

\[
= \Pr[\text{res} \leftarrow G : \text{res} \land \langle e \rangle_C \notin L_H] + \Pr[\text{res} \leftarrow G : \text{res} \land \langle e \rangle_C \in L_H]
\]

\[
= \Pr[\text{CPA}_{PG,A}(c_2, \phi_2 \land \text{Ask}(H,e)) = \text{true}] + \Pr[\text{res} \leftarrow G : \text{res} \land \langle e \rangle_C \in L_H]
\]

\[
\leq \Pr[\text{CPA}_{PG,A}(c_2, \phi_2) = \text{true}] + \Pr[\text{res} \leftarrow G : \text{res} \land \langle e \rangle_C \in L_H]
\]

\[
= \Pr[\text{CPA}_{PG,A}(c_3, \phi_3) = \text{true}] + \Pr[\text{CPA}_{PG,A}(c_3, \phi_3) = \text{true}]
\]

The first step follows from \( c_1^* = c_2^* \{H(e)/r\} \) and \( \phi_1 = \phi_2 \{H(e)/r\} \). The second steps follows from the inclusion-exclusion principle. The third step exploits that \( G \) and \( \text{CPA}_{PG,A}(c_2, \phi_2) \) are equivalent until the adversary queries \( \langle e \rangle_C \) to \( H \). In the fourth, we weaken the event in the first probability. The fifth step follows from \( c_3^* = c_2^* \{H(e)/r\} \) and \( \phi_3 = \phi_2 \{H(e)/r\} \land \text{Ask}(H,e) \).

**Terminal Rules:** We have three such rules: \([\text{Ind}], [\text{Rnd}] \) and \([\text{OW}] \). The first two follow by an information theoretic argument whereas the latter follows by a computational reduction to the one-wayness of the underlying trapdoor permutation scheme. In all three cases, we must prove that

\[
\Pr[[c^* : \phi]_{PG,A} = \Pr[\text{CPA}_{PG,A}(c^*, \phi) = \text{true}]] \leq B_{(t_A, \delta)}(\nabla).
\]

for all adversaries \( A \) executing within time \( t_A \) and performing at most \( \delta \) queries. In the proofs, we use \( \langle C \rangle(p) \) to denote the program \( \square \leftarrow p; \langle C \rangle \) for a context \( C \) and a program expression \( p \).

**Case** \([\text{Ind}] \). We know that \( m \notin \mathcal{X}(c^*) \). Therefore, the randomly sampled bit \( b \) is independent of the view of the adversary and we conclude that

\[
\Pr[\text{CPA}_{PG,A}(c^*, \phi) = \text{true}] = \frac{1}{2} = B_{(t_A, \delta)}(\nabla).
\]

**Case** \([\text{Rnd}] \). Let \( \phi = \text{Ask}(H_1, e_1) \land \ldots \land \text{Ask}(H_n, e_n) \) and \( C \{e_1 \ldots e_n/\square\} = E r_1 \ldots r_k \) for some context \( C \) such that \( \vert r_1 \vert + \ldots + \vert r_k \vert = \ell \) and \( r_i \notin \mathcal{R}(c^*) \) for \( i \in \{1, \ldots, k\} \). We conclude the proof as follows:

\[
\Pr[\text{CPA}_{PG,A}(c^*, \phi) = \text{true}]
\]

\[
= \Pr[\text{CPA}_{PG,A}(c^*, \phi) : \bigwedge_{i=1}^{n} \langle e_i \rangle_C \in L_{H_i}]
\]

\[
= \Pr[\text{CPA}_{PG,A}(c^*, \phi) : \bigvee_{i_1=1}^{L_{H_1}} \ldots \bigvee_{i_n=1}^{L_{H_n}} \langle e_1 \rangle_C = L_{H_1}[i_1] \land \ldots \land \langle e_n \rangle_C = L_{H_n}[i_n]]
\]
Chapter 2. Automated analysis of padding-based encryption

\[ \Pr [ \text{CPA}_{PG, A}(c^*, \phi) : \bigvee_{i_1=1}^{|L_{H_1}|} \ldots \bigvee_{i_n=1}^{|L_{H_n}|} (C)_C(L_{H_1}[i_1] \ldots L_{H_n}[i_n]) = r_1 \ldots r_k ] \]  

(3)

\[ \leq \sum_{i_1=1}^{L_{H_1}} \ldots \sum_{i_n=1}^{L_{H_n}} \Pr [ \text{CPA}_{PG, A}(c^*, \phi) : (C)_C(L_{H_1}[i_1] \ldots L_{H_n}[i_n]) = r_1 \ldots r_k ] \]  

(4)

\[ \leq \sum_{i_1=1}^{L_{H_1}} \ldots \sum_{i_n=1}^{L_{H_n}} 2^{-\ell} \]  

(5)

\[ \leq \prod_{i=1}^{n} q_{H_i} \times 2^{-\ell} \]  

(6)

\[ = B_{(1_{A}, \phi)}(\nabla) \]

The first step unfolds the semantics of the event \( \phi \). The second step rewrites the event with disjunctions. In the third step, and exploit that \( C \{e_1\ldots e_n/\overline{c}\} = E r_1 \ldots r_k \). The fourth step uses the union-bound and the fifth step exploits that \( r_1 \ldots r_k \) is a uniformly sampled bitstring of length \( \ell \) that is independent of the view of the adversary. The final step simplifies the expression and uses the bounds on the number of queries performed by the adversary.

**Case** \( \text{OW}_1(C, C') \). Let \( \phi = \text{Ask}(H_1, e_1) \land \ldots \land \text{Ask}(H_n, e_n) \) and consider the inverter

**Procedure** \( \mathcal{I}_A(y, f) \) \[\text{def}\] sample function for each \( H \in \mathcal{H} \)

\((m_0, m_1, \sigma) \leftarrow A^{B}_f (f) ; b \leftarrow \{0, 1\} ; m \leftarrow m_b ; \)

sample bitstring for each \( r \in \bar{r}_2 \)

c \[\leftarrow \{C\}_C(y \parallel r_2 \parallel m) ; b \leftarrow A^{B}_f (c^*, \sigma) ; \)

\( S \leftarrow \{(C)_C(x_1 \ldots x_n \parallel r_2 \parallel m) \mid x_1 \in L_{H_1} \land \ldots \land x_n \in L_{H_n} \}; \)

return \( S \);

which allows us to conclude the proof as follows:

\[ \Pr [ \text{CPA}_{PG, A}(c^*, \phi) = \text{true} ] = \Pr [ \text{OW}(\mathcal{I}_A) : (e_1)_C \in L_{H_1} \land \ldots \land (e_n)_C \in L_{H_n} ] \]  

(1)

\[ \leq \Pr [ \text{OW}(\mathcal{I}_A) : [\bar{r}_1]^\ell_k \in S \land |S| \leq \prod_{i=1}^{n} q_{H_i} ] \]  

(2)

\[ \leq \text{Succ}^{s-pd-\text{OW}}(k, \ell, \prod_{i=1}^{n} q_{H_i}, t_L) \]  

(3)

\[ = B_{(1_{A}, \phi)}(\nabla) \]  

(4)

The first step exploits that after inlining \( \mathcal{I}_A \) in \( \text{OW} \), the two games are equivalent for a one-way challenger \( \text{OW} \) that samples \( f \) and \( \bar{r}_1 \). In the second step, we use that \( (e_1)_C \in \)
2.7. Proofs

\( L_{H_1} \land \ldots \land L_{H_n} \) implies that \([r_i]^t_k \in S\) since \((C)\_C(x_1 || \ldots || x_n || r_2 || m_b) = [r_i]^t_k\) for \(x_i = (e_i)_C\). This implies that \(\mathbb{I}_A\) inverts \(f\) on the bits \([k \ldots k+l-1]\) in the game \(\text{OW(}\mathbb{I}_A\text{)}\). Since \(\mathbb{I}\) runs in time \(t_\mathbb{I} = t_A + T(C') + \prod_{i=1}^{n} q_{H_i} \times T(C)\), we can therefore conclude the proof with the third and the fourth step. \(\blacksquare\)

2.7.2 Soundness of CCA logic

We now define an extended CPA game to state and prove the soundness of the extended CPA logic used in the CCA logic.

Definition 2.6 (Extended CPA game). Let \(\mathcal{P}G\) be a permutation generator, \(\mathbb{A}\) an adversary, \(c^*\) an expression and \(\phi\) an event of the extended CPA logic, i.e. including events of the form \(\exists x \in \mathbb{L}_D, \phi\), then we define the extended generic CPAE game as follows

\[
\begin{align*}
\text{Game } \text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) \ &= \text{def} \\
L_1 & \text{ sample function for each } H \in \mathcal{H} \\
L_2 & (f, f^{-1}) \leftarrow \mathcal{P}G; \\
L_3 & (m_0, m_1, \sigma) \leftarrow A_1^\mathcal{P}(f); \\
L_4 & b \leftarrow \{0, 1\}; m \leftarrow m_b; \\
L_5 & \text{ sample bitstring for each } r \in \mathcal{R}(c^*, \phi) \\
L_6 & \bar{b} \leftarrow A_2^\mathcal{P}(\langle c^* \rangle_C, \sigma); \\
L_7 & \text{ return } \langle \phi \rangle_C
\end{align*}
\]

where \(A_1\) and \(A_2\) are a pair of procedures representing \(\mathbb{A}\). Overloading notation, we define

\[
\Pr[\|c^* : \phi\|_{\mathcal{P}G,\mathbb{A}} = \Pr[\text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) = \text{true}]\\
\text{for the CPAE logic.}
\]

Lemma 2.5. The CPAE logic remains sound in the sense of Theorem 2.1.

Proof. Rather than repeating the proof of Theorem 2.1 in the extended setting, we explain how the proof is adapted. Note that the justification of the bridging rules, hash rules and the \([\text{Ind}]\) can be adapted in a straightforward way.

For the remaining terminal rules, we have events of the form \(\phi = \exists c. \text{Ask}(H_1, e_1) \land \ldots \land \text{Ask}(H_n, e_n)\) and a context \(C\) such that \(C[e_1 || \ldots || e_n] = E [r_i]^t_k\). In order to adapt these proofs we reason as follows

\[
\begin{align*}
\Pr[\|c^* : \phi\|_{\mathcal{P}G,\mathbb{A}} \\
\ &= \Pr[\text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) = \text{true}] \\
\ &= \Pr[\text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) \land \exists c \in \mathbb{L}_D. \text{Ask}(H_1, e_1) \land \ldots \land \text{Ask}(H_n, e_n)] \\
\ &= \Pr[\text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) \land \exists \bar{x} \in \mathbb{L}_H. \exists c \in \mathbb{L}_D. \bar{x} = \langle c \rangle_C] \\
\ &\leq \Pr[\text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) \land \exists \bar{x} \in \mathbb{L}_H. \exists c \in \mathbb{L}_D. \bar{x} = \langle c \rangle_C \land \langle C[\bar{x}] \rangle_C = [r_i]^t_k] \\
\ &\leq \Pr[\text{CPAE}_{\mathcal{P}G,\mathbb{A}}(c^*, \phi) \land \exists \bar{x} \in \mathbb{L}_H. \langle C[\bar{x}] \rangle_C = [r_i]^t_k]
\end{align*}
\]
where \( c^* \) is an expression. The first two steps follow by definition, the third one by logical equivalence, the fourth step follows from the fact that \( =_E \) is sound and hypothesis, the fifth step follows from logical implication and the last step follows from the fact that \( c \notin \mathcal{X}(C[\vec{x}]) \).

Note that we have eliminated the existential quantification ranging on the \( L_D \) list because \( c \) is not really needed by \( C \). This justifies the fact that the bound for these terminal rules is not scaled with \( q_D \). The remaining part of the proofs follows the same reasoning as their CPA counterpart.

The terminal rule \([\text{Eqs}(r)]\) is proved in a similar way as \([\text{Rnd}]\) for CPA, relying on the fact that the adversary triggers an event that entails guessing randomly sampled values \( r_i \) independent of its view, since \( r_i \notin \mathcal{R}(c^*) \).

\[ \mathbf{Lemma 2.6.} \text{ Let } T \text{ be a test (an boolean valued expression) and } \nabla \text{ a derivation of } t_0 \models T, \text{ then for any command } c \text{ that sets } f \text{ and } f^{-1} \text{ as two permutations inverse of eachother, any time } t \text{ and any bound on oracle calls } q \text{ we have that} \]

\[
\Pr \left[ c; r \not\models \{0,1\}^{|r|} : \langle T \rangle_C \right] \leq B(t,\bar{q})(\nabla)
\]

\[ \text{Proof. } \text{The proof follows by induction on the derivation } \nabla. \]

For \([\text{PAnd}]\) we have

\[
\Pr \left[ c; r \not\models \{0,1\}^{|r|} : \langle T_1 \land T_2 \rangle_C \right] \leq \Pr \left[ c; r \not\models \{0,1\}^{|r|} : \langle T_1 \rangle_C \land \langle T_2 \rangle_C \right] \\
\leq \Pr \left[ c; r \not\models \{0,1\}^{|r|} : \langle T_2 \rangle_C \right] \\
\leq \sum_{i=0}^{q_H} \Pr \left[ c; r \not\models \{0,1\}^{|r|} : C[L_H[i]] = \{r\}^{|r|}_k \right] \\
\leq q_H \times 2^{-\ell}.
\]

The remaining rule is proved in the same way, but there is no need for the scaling factor \( q_H \).

\[ \mathbf{Definition 2.7 (Semantics of Find).} \text{ Let } \vec{x} \text{ be a sequence of variables, } L_H \text{ a sequence of hash oracles, } T \text{ a test and } t \text{ an expression then we define} \]

\[
\langle \text{find } \vec{x} \text{ in } L_H : T \triangleright \theta \rangle_C = \text{pick}_\perp \{\theta\}_C \mid \vec{x} \in L_H \land \langle T \rangle_C
\]
2.7. Proofs

where \( \text{pick}_\bot \) returns an arbitrary element of the argument set or \( \bot \) when the it is empty.

**Definition 2.8** (Generic CCA game). Let \( PG \) be a trapdoor permutation generator, \( A \) a CCA adversary, \( c^* \) an expression, \( D \) a decryption algorithm, and \( \phi \) an event, then we define the game \( \text{CCA}_{PG,A}(c^*,D,\phi) \) exactly as \( \text{CPA}_{PG,A}(c^*,\phi) \), except that the adversary can query a decryption oracle implemented by evaluating \( \langle D \rangle_C \) up to \( q_D \) times. We keep track of the decryption queries during the second phase using the list \( L_D \) and we ignore queries for \( c^* \) in the second phase. Analogously to the CPA logic, we also define

\[
\Pr \|(c^*, D) : \phi\|_{PG,A} = \Pr \left[ \text{CCA}_{PG,A}(c^*, D, \phi) = \text{true} \right].
\]

**Theorem 2.3.** Let \( c^* \) be an expression, \( D \) a decryption oracle, \( \phi \) an event and \( \hat{\phi} \) a probability tag. If \( \nabla \) is a derivation of \( \models_{\hat{\phi}} (c^*, D) : \phi \) under \( \Gamma \), then

\[
B_{(t_A, \hat{\phi})}(\nabla) \geq \begin{cases} 2 \Pr \| (c^*, D) : \phi \|_{PG,t_A,\hat{\phi}} - 1 & \text{if } \phi = \text{Guess} \\ \Pr \| (c^*, D) : \phi \|_{PG,t_A,\hat{\phi}} & \text{otherwise} \end{cases}.
\]

**Proof.** Again, we distinguish between bridging rules, terminal rules and [Reject].

**Bridging Rules** It suffices to show that the probability of the judgement in the conclusion is upper bounded by that in the premise. [Conv] is proved by reasoning as in [Sub]. For [Find], we proceed by reasoning algebraically on the sets denoted by the algorithms as part of the semantics of finds and noting that \( \text{Ask}(H,e) \) is equivalent to \( \exists x \in H.x = e \).

**Terminal Rules** We have two terminal rules: [False] and [\( \text{Pub}(F,T,t) \)].

The former follows from the fact that \( c^* \not\in L_D \) is an invariant, therefore the event \( \exists c \in L_D.c = c^* \) has probability 0.

For [\( \text{Pub}(F,T,t) \)], note that since the decryption oracle is public, i.e. it doesn’t use \( f^{-1} \) the adversary can simulate it himself. Therefore, we first prove that for any adversary \( A \) against CCA, there exists an adversary \( S \) against the extended CPA game such that

\[
\Pr \| (c^*, D) : \phi \|_{PG,A} \leq \Pr \| c^*: \phi \|_{PG,S}.
\]

We construct \( S \) as a pair of functions \( S_1 \) and \( S_2 \); and a decryption oracle \( D \)

\[
D(\hat{c}) = L_D \leftarrow \hat{c} :: L_D; \text{return } \langle D \{ \hat{c}/c \} \rangle_C;
\]

\[
S_1^{\text{H}}(f) = (m_0,m_1) \leftarrow A_1^{\text{H},D}(f); \text{return}(m_0,m_1);
\]

\[
S_2^{\text{H}}(c^*) = \bar{b} \leftarrow A_2^{\text{H},D}(c^*); \text{return}(\bar{b},L_D);
\]

where \( D = F : T \triangleright t \).

Moreover, we note that the time and hash calls to \( H_t \) by \( S \) are defined as follows

\[
t_S = t_A + q_D \times (T(t) + T(T) \times \prod_{H_j \in F} q_{H_j})
\]

\[
q_{H_t}^S = q_{H_t} + q_D \times (Q_t(t) + Q_t(T) \times \prod_{H_j \in F} q_{H_j})
\]
where $T(e)$ and $Q_i(e)$ denote respectively the time and the hash calls to $H_i$ taken to compute $e$ and $t_A$ and $q_B$ are the time and the calls to $H_i$ used adversary $A$.

Therefore we can conclude that for any time $t_A$ and resource bound $q$, we have

$$\Pr[(e^*, D) : \phi]_{PG, t_A, q} \leq \Pr[e^* : \phi]_{PG, t_S, q^S}.$$ 

where $t_S$ and $q^S$ are defined as above.

We conclude by noting that from Lemma 2.5, we know that

$$\Pr[e^* : \phi]_{PG, t_S, q^S} \leq B(t_S, q^S)(\nabla)$$

where $\nabla$ is a derivation of $\equiv_t e^* : \phi$, such as the premise of the rule.

[Reject] By inductive hypotheses, we have

$$\Pr[(e^*, D_2) : \phi]_{PG, A} \leq B(t_A, q)(\nabla_1)$$
$$\Pr[(e^*, D_2) : \exists c \in L_D.T \wedge e = e^*]_{PG, A} \leq B(t_A, q)(\nabla_2)$$
$$\Pr[r' \neq \{0, 1\}^{\lfloor r \rfloor} : T\{r'/H(e)\}] \leq B(t_A, q)(\nabla_3)$$

where $\nabla_1$ is a derivation of $\equiv_{\hat{b}} (e^*, D_2) : \phi$, $\nabla_2$ is a derivation of $\equiv_{\hat{b}} (e^*, D_2) : \exists c \in L_D.T \wedge e = e^*$ and $\nabla_3$ is a derivation of $\equiv_0 r' \neq \{0, 1\}^{\lfloor r \rfloor} : T\{r'/H(e)\}$, each corresponding to premises of the rule.

Consider the two following decryption algorithms $D_1 = T \triangleright t$ and $D_2 = T \triangleright \neg \text{Ask}(H, e) \triangleright t$. These two yield different results if the argument $\hat{c}$ is such that $T\{c/\hat{c}\} \wedge \neg \text{Ask}(H, e)$. If an adversary distinguishes the two, then we know that $\exists c \in L_D.T \wedge \neg \text{Ask}(H, e)$. Therefore, we have that for any CCA adversary $A$

$$\Pr[(e^*, D_1) : \phi]_{PG, A}$$
$$= \Pr[\text{CCA}_{PG, A, e^*, D_1, \phi} : \text{res}]$$
$$\leq \Pr[\text{CCA}_{PG, A, e^*, D_2, q} : \text{res}] + \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge \neg \text{Ask}(H, e)} : \text{res}]$$
$$= \Pr[(e^*, D_2) : \phi]_{PG, A} + \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge \neg \text{Ask}(H, e)} : \text{res}]$$
$$\leq B(t_A, q)(\nabla_1) + \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge \neg \text{Ask}(H, e)} : \text{res}] .$$

Now, we perform further case distinction on whether the expression $e$ was queried to $H$ during the construction of the cipher $e^*$. If this is the case, then $e = e^*$, where $e^*$ is the argument of the hash $H$ in $e^*$. If this is not the case, then $H(e)$ is a fresh query and therefore, indistinguishable from a random value $r'$ and we have

$$\Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge \neg \text{Ask}(H, e)} : \text{res}]$$
$$\leq \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge e = e^*} : \text{res}] + \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \times \{r'/H(e)\}} : \text{res}]$$
$$= \Pr[(e^*, D_2) : \exists c \in L_D.T \wedge e = e^*]_{PG, A} + \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge \{r'/H(e)\}} : \text{res}]$$
$$\leq B(t_A, q)(\nabla_2) + \Pr[\text{CCA}_{PG, A, e^*, D_2, \exists c \in L_D.T \wedge \{r'/H(e)\}} : \text{res}] .$$

where $\nabla_2$ is a derivation of $\equiv_{\hat{b}} (e^*, D_2) : \exists c \in L_D.T \wedge e = e^*$. Finally, note that the remaining event corresponds to the test $T\{r'/H(e)\}$ holding when $r'$ is randomly sampled.
This property is captured succinctly by the judgement $\vdash r' \notin T \{r'/H(e)\}$. We bound it by reasoning as follows

$$
\Pr \left[ \text{CCA}_{P_G,A,c^*,D_2,\exists c \in L_D}.T \{r'/H(e)\} : \text{res} \right] = \Pr \left[ \text{CCA}_{P_G,A,c^*,D_2,\text{true}} : \exists c \in L_D.T \{r'/H(e)\} \right] \\
\leq \sum_{i=0}^{n} \Pr \left[ \text{CCA}_{P_G,A,c^*,D_2,\text{true}} : T \{r'/H(e)\} \{L_D[i]/c\} \right] \\
\leq q_D \times \Pr \left[ r' \notin \{0,1\}^{r} : T \{r'/H(e)\} \right] \\
\leq q_D \times B(t_{A,\vec{q}})(\nabla_3).
$$

Putting everything together we obtain the desired property

$$
\Pr \left[ (c^*, D_1) : \phi \right]_{P_G,A} \leq B(t_{A,\vec{q}})(\nabla_1) + B(t_{A,\vec{q}})(\nabla_2) + B(t_{A,\vec{q}})(\nabla_3).
$$
Chapter 2. Automated analysis of padding-based encryption
Cryptographic protocols, like TLS, SSH, and VPNs, are one of the main building blocks of the Internet. At the heart of these protocols lies a key exchange protocol, which allows two parties to establish a shared session key used for building a secure channel. Traditionally, key exchange has often been realized using key transport protocols. Here, one participant generates the session key and uses public key encryption and signatures to transport it to the peer. Since this approach usually uses a longterm public key for encryption, it lacks resilience against leakage of the corresponding secret key, either through cryptanalysis or coercion. Concretely, if an adversary obtains the longterm secrets of a participant, he can obtain all his session keys. Resilience against such attacks is called forward secrecy [55]. While long known in the cryptographic community, forward secrecy has recently come under public light following revelations about mass surveillance and implementation bugs such as Heartbleed. As a consequence, we expect that the ongoing shift from key transport protocols to key agreement protocols that achieve forward secrecy will accelerate; for instance, there is consensus to deprecate RSA key transport in TLS 1.3.

One solution to achieve forward secrecy is to use protocols that use an ephemeral Diffie-Hellman (DH) exchange. Since the ephemeral DH exchange uses fresh exponents for each session, protocols using them can provide forward secrecy. In order to provide authentication, most popular protocols such as TLS and SSH sign the exchanged DH messages. Theoretically, key agreement protocols based on signed DH are well understood and allow for relatively straightforward proofs of the classical security properties and forward secrecy [28, 42]. In practice, their usage in real-world protocols poses additional problems and there is a large body of work on analyzing the security of the combined channel establishment protocol [67, 72, 33].

Nevertheless, the use of signatures has several disadvantages. First, standardization and implementation must include a signature scheme which might not be required otherwise. Second, the use of signatures might compromise deniability [70]. Third, signing and
verification time might be a bottleneck. Furthermore, several realistic attacks are still possible for one-round versions of such protocols. For example, leakage of session randomness can lead to the compromise of future sessions in signed DH protocols [71, Section 1.6].

To address these deficiencies, implicitly authenticated key exchange (IAKE) protocols have been introduced in [82]. Such protocols enhance an ephemeral DH exchange with static DH keys that are only used in the key computation. Many protocols of this type have been proposed, such as HQMV [71], Naxos [76], and Nets [78], and they often surpass signature-based protocols in terms of performance and security. For example, the HMQV protocol, which is a hashed variant of the MQV [77] protocol, adds authentication to the ephemeral Diffie-Hellman protocol at a very low cost if Shamir’s trick [61] is used for multi-exponentiation. Prominent instances of deployed systems based on such protocols include the EMV [41] chip based payment system, which uses a custom protocol and Blackberry phones, which use the elliptic curve version of MQV [77]. One of the main adversary models for IAKE protocols is the extended Canetti-Krawczyk (eCK) model [76], which provides very strong security guarantees such as (weak) perfect forward secrecy and session key secrecy even in the case where the session’s randomness is leaked.

However, a number of concerns with the provable security of this class of protocols remain. First, only some of them achieve efficient designs and tight reductions under standard assumptions such as computational DH (CDH). Instead, known proofs of efficient protocols often use the Forking Lemma (and therefore give non-tight reductions), or strong assumptions such as Gap-CDH [87]. Second, and probably more importantly, the security definitions for key exchange protocols are an order of magnitude more complex than standard definitions for most cryptographic primitives, such as IND-CCA. This results in long proofs that few people understand or check for flaws. Unsurprisingly, numerous attacks have been discovered on key exchange protocols [68, 71, 83, 84], even on those claimed provably secure. This second problem is not exclusive to key exchange protocols. In fact, two approaches have been developed to tame the complexity of cryptographic proofs in the computational model.

The first approach is to develop generic results that can be applied to many concrete instances. While genericity does not eliminate the possibility of flaws, it allows to build a reduced corpus of results on which the security of protocols depends, and gives greater incentive to examine their proofs carefully. One popular class of generic results in cryptography are protocol transformations. If a protocol $\Pi$ is secure with respect to an adversary model $\mathcal{M}$, then $\Pi$ can be transformed into a (more complicated) protocol $\Pi'$ that is secure with respect to a stronger adversary model $\mathcal{M}'$. For key exchange, this approach was pioneered by Bellare, Canetti, and Krawczyk [26] and other transformations have been proposed by Kudla and Paterson [73], Cremers and Feltz [53], and Boyd et al. [40]. However, existing transformations have several drawbacks, in particular: the transformation in [26] cannot be applied to many protocols of interest; the transformations in [53, 40] assume that the initial protocol is already secure in the eCK model; and the transformation in [73] only supports proofs under Gap assumptions, predates the eCK model and is only applicable to weaker security models.
The second approach is to build machine-checked, independently verifiable proofs of security; this approach has been suggested notably by Halevi [64], and more recently by Hales\footnote{https://jiggerwit.wordpress.com/2013/11/04/formalizing-nist-standards/} in the context of verifying the absence of trapdoors in NIST standards. Assuming that the verification tool is correct, one can gain trust in a formal proof simply by checking the definitions it uses and the theorem statement, since the tool ensures the correctness of the reasoning steps. There are two mature tools to perform machine-checked cryptographic proofs in the computational model: CryptoVerif [35] and EasyCrypt [13, 10]. CryptoVerif is an automatic prover in the computational model and has been applied to cryptographic constructions such as the Full Domain Hash signature scheme, Kerberos, and the One-Encryption Key Exchange. EasyCrypt is a toolset for the construction and verification of game-based cryptographic proofs and has mostly been applied to cryptographic primitives, such as the Cramer-Shoup encryption scheme, and the OAEP padding scheme. So far neither of these tools have been used to obtain machine-checked proofs of modern key exchange protocols with respect to their intended security definitions.

Both approaches are complementary. Indeed, machine-checked proofs make checking proofs efficient, but they also significantly increase the cost of building proofs. As a consequence, generic results are ideal targets for machine-checked proofs, for two reasons. First, the cost of building proofs for generic results is justified by their multiple applications. Second, the level of abstraction required to obtain generic proofs combined with the explicit tracking of assumptions in machine-checked proofs often provides new insights.

Contributions

We develop a new generic proof of security for key-exchange protocols, and instantiate it to obtain security proofs for known protocols with respect to different adversary models and hardness assumptions. In the cases of Naxos and Nets, we show that it is possible to obtain a CDH proof (without GAP) if static keys are honestly generated. We also formalize our generic proof and its instantiations using EasyCrypt. We elaborate on these points below.

**Generic Proof for eCK security.** We consider the class of one-round Diffie-Hellman protocols defined in the random oracle model where the session key is the output of a hash function. We reduce eCK-security of a key exchange protocol in this class to a condition on the key computation function and four simple games, in which the adversary can access at most one oracle. For protocols that employ the Naxos trick and use \( h(x, a) \) as the exponent of the DH message, we provide an even simpler reduction with three games.

Concretely, we structure our generic proof in terms of protocol transformations and different versions of the security game. We are interested in eCK security. As proof tools, we also use three additional security games:

- eCK: Adversary must distinguish the session key of a fresh test session from random key.
Chapter 3. Modular Proofs of Key Exchange Protocols

- **eCK\textsuperscript{nt}:** Variant of eCK where adversary must provide the actor’s static secret key as input to the ephemeral reveal oracle.

- **CSK:** Simplified game for protocols that do not use the Naxos trick where adversary must compute session key of test session (4 cases).

- **CSK\textsuperscript{nt}:** Simplified game for protocols that use the Naxos trick where adversary must compute session key of test session (3 cases).

We then define protocol transformations $T\textsuperscript{nt}$ (use Naxos trick) and $T\textsuperscript{hsk}$ (hash session key) and prove that the following implications hold for all protocols $\Pi$:

- $\Pi$ is eCK\textsuperscript{nt}-secure $\implies T\textsuperscript{nt}(\Pi)$ is eCK-secure
- $\Pi$ is CSK\textsuperscript{nt}-secure $\implies T\textsuperscript{hsk}(\Pi)$ is eCK\textsuperscript{nt}-secure
- $\Pi$ is CSK-secure $\implies T\textsuperscript{hsk}(\Pi)$ is eCK-secure

As an example, consider the Naxos protocol which uses the Naxos trick and hashes its session key. We first define the ‘core’ of Naxos and call it $\text{Naxos}\text{core}$. Since it holds that $\text{Naxos} = T\text{hsk}(T\text{nt}(\text{Naxos}\text{core}))$, it suffices to prove that $\text{Naxos}\text{core}$ is CSK\textsuperscript{nt}-secure to obtain that Naxos is eCK-secure. While the original eCK security definition consists of a game with seven oracles where the winning condition contains a complicated freshness condition, the CSK\textsuperscript{nt} game has a very simple winning condition and only provides a decision oracle that allows the adversary to confirm session key guesses.

To compare different models of key distribution, we support two versions of the eCK model: The eCK\textsubscript{nkr} model where all static keys are honestly generated and the eCK\textsubscript{kr} model that allows the adversary to adaptively register arbitrary public keys for dishonest parties without providing a proof of possession. The original eCK model [76] sits in between our two versions. The adversary can register arbitrary public keys for dishonest parties before activating the first session, i.e., the registered public keys can depend on public keys of honest parties, but not on protocol messages, as, for example, required for Kaliski’s attack [68] on MQV.

Our proof improves [73] in several ways: it uses the stronger eCK adversary model (with and without adversarial key registration); it supports proofs under standard assumptions (whereas the proof from [73] requires Gap assumptions), and; it exploits the Naxos trick resulting in simpler proof obligations for protocols that use it.

**Concrete Proofs.** We instantiate the generic proof to obtain security proofs for existing protocols; in all cases, the proofs of the simplified games are short by the standards of machine-checked proofs. Our results are summarized in Table 3.1. Concretely, we prove that:

- Naxos and Nets are secure in the eCK model under the CDH assumption if keys are honestly generated. If we allow arbitrary adversarial key registration, we require the Gap-CDH assumption as in the original proof.
3.1. Background

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Existing Proof</th>
<th>Our Proofs</th>
<th>EasyCrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naxos [76]</td>
<td>eCK/Gap-CDH</td>
<td>eCKkr/Gap-CDH, eCKnkr/CDH</td>
<td>yes</td>
</tr>
<tr>
<td>Nets [78]</td>
<td>eCKkr/Gap-CDH</td>
<td>eCKkr/Gap-CDH, eCKnkr/CDH</td>
<td>yes</td>
</tr>
<tr>
<td>Naxos+ [79]</td>
<td>eCKkr/Gap-CDH</td>
<td>eCKkr/CDH</td>
<td>yes</td>
</tr>
<tr>
<td>HMQV [71]</td>
<td>CKIMQV/Gap-CDH+KEA1</td>
<td>eCKkr/Gap-CDH</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 3.1: Obtained proofs for Key Exchange Protocols (*see explanation, nt=non-tight).

- The Naxos variant Naxos+ [79] is secure in the eCK model with arbitrary adversarial key registration under the CDH assumption. Here we obtain a similar result to the original proof using our generic proof method.

- A version of HMQV is secure in the eCK model under the Gap-CDH assumption. The version we analyze includes the identities and exchanged message in the input of the key derivation hash. The proof does not need KEA1 (knowledge of exponent assumption).

**EasyCrypt Formalization.** We have formalized all models, our generic proof for protocols using the Naxos trick, and the proofs for Nets, Naxos, and Naxos+ in EasyCrypt. Our formalization constitutes the biggest case study developed with the tool so far; e.g. the generic proof for protocols using the Naxos trick takes about 30,000 lines of code, including game definitions (about 50 of them), specifications, and proofs. On the other hand, the instantiation of the proof for concrete protocols is short and takes less than 1,000 lines each. Our formalization also includes several reusable libraries that deal with random oracles, Twin DH, and common proof techniques such as plug and pray, that we discuss in Section 3.1.3.

3.1 Background

In this section, we give some background on notation, AKE protocols, and EasyCrypt.

3.1.1 Notation

$A^*$ denotes the set of all sequences with elements taken from $A$. For two sequences $s_1$ and $s_2$, we use $S_1 \leftrightarrow S_2$ to denote their concatenation. We use $s_1 \leftarrow s_2$ as a shorthand for the assignment $s_1 \leftarrow s_1 \leftrightarrow s_2$. In the special case of bitstrings $b_1$ and $b_2$, we also use $b_1b_2$ to denote their concatenation.

We use $A \to B$ to denote the set of partial functions from $A$ to $B$. If $f$ is a (partial) function, we define $f[a := b]$ as the function $x \mapsto \text{if } x = a \text{ then } b \text{ else } f(x)$. In games, we use $f[a] \leftarrow b$ as a shorthand for $f \leftarrow f[a := b]$ (update $f$ at key $a$). For a finite set $A$, we use $x \leftarrow A$ to denote that $x$ is uniformly sampled from $A$.

We use $\mathbb{G}$ to denote a cyclic group of prime order $p$ with generator $g$. We use $\mathbb{F}_p$ to denote the field of integers modulo $p$. We use $\text{dlog}(Y)$ to denote the discrete logarithm.

55
of $Y$ with respect to the basis $g$. We define $\text{cdh}(X,Y) \triangleq X^{\text{dlog}(Y)}$ and $\text{ddh}(X,Y,Z) \triangleq (\text{cdh}(X,Y) = Z)$. Based on the previous definitions, we define the following cryptographic assumptions. The challenger for DLOG gives $X \leftarrow \mathbb{G}$ to the adversary who must return $\text{dlog}(X)$. The challenger for CDH gives $X,Y \leftarrow \mathbb{G}$ to the adversary who must return $\text{cdh}(X,Y)$. For SCDH, the adversary is given the same challenge, but must return a set containing $\text{cdh}(X,Y)$. We also define Gap versions [87] of these assumptions where the adversary is given access to an oracle that returns $\text{ddh}(X,Y,Z)$ for arbitrary $X,Y,Z \in \mathbb{G}$.

### 3.1.2 One-Round Authenticated Key Exchange Protocols

In the following, we focus on one-round key exchange protocols. We believe most of our results can be extended to a more general notion of protocol. Further note that our results are not restricted to DH-based protocols and the formal definitions in Section 3.2.1 will generalize some of the notions we introduce informally in this section.

Figure 3.1 shows the computations and exchanged messages for a typical DH-based key exchange protocol. We assume a protocol consists of three components. First, there is a protocol component for key generation, which we show in the first line. Here, a participant $A$ samples the static secret key $a$ and computes the corresponding static public key $A$. Second, there is a component responsible for the distribution of the static public keys. We ignore the details for now and just assume that an agent can obtain the public key of another agent.

Finally, there is a component responsible for establishing the session key. This component consists of an initiator role and a responder role. If an agent $A$ executes an instance of the initiator (resp. responder) role with the goal of establishing a session key with $B$, we call this execution a session with role initiator (resp. responder), actor $A$, and peer $B$.

When the initiator role is activated with actor $A$ and peer $B$, it first generates an ephemeral secret key $x$, computes the ephemeral public key $X$, sends it to $B$, and waits for a reply. When the responder $B$ is activated with a received message from $A$, he stores the received message as $X$, generates $y$ and $Y$ in the same way as the initiator, sends $Y$ to $A$, and computes the session key using the $\text{KeyR}$ function. When the initiator is activated with the received message, he computes the session key using $\text{KeyI}$.

We can define the HMQV protocol by using $\text{KeyI}/\text{KeyR}$ that compute the key as $H(\sigma)$ for $\sigma = \text{cdh}(X,Y)\text{cdh}(X,B)^{\epsilon}\text{cdh}(A,Y)^{d}\text{cdh}(A,B)^{de}, e = h(X, B)$, and $d = h(Y, A)$. We can define the Naxos and Nets protocols by using the boxed expressions from Figure 3.1 to compute $X$ and $Y$. These protocols both utilize the Naxos trick which combines the static and the ephemeral secret using the hash function $h$ to obtain the exponent of ephemeral public key. Since the hash output is never stored and recomputed when required, these protocols are analyzed with respect to possible leakage of $x$ or $a$, but leakage of $h(x, a)$ is not considered. The Naxos protocol defines the session key as $H(\sigma)$ for $\sigma = \text{cdh}(A,Y)\text{cdh}(X,B)\text{cdh}(X,Y)AB$. The Nets protocol defines the session key as $H(\sigma)$ for $\sigma = \text{cdh}(X,Y)\text{cdh}(X,B)\text{cdh}(A,Y)\text{cdh}(A,B)\text{cdh}(X,Y)ABXY$. The Naxos+ protocol extends the Naxos protocols with the additional group element $\text{cdh}(A,B)$, i.e., the session
3.1. Background

\[
\begin{align*}
\hat{A} : a &\leftarrow \mathbb{F}_p, A = g^a \\
\hat{B} : b &\leftarrow \mathbb{F}_p, B = g^b \\
x &\leftarrow \mathbb{F}_p \quad g^x \quad g^{h(x,a)} \quad \text{receive } X \\
\text{receive } Y &\leftarrow \mathbb{F}_p \quad g^y \quad g^{h(y,b)} \quad y \leftarrow \mathbb{F}_p
\end{align*}
\]

Figure 3.1: Generic two pass protocol. Protocols using the Naxos trick use boxed messages.

key is defined as \(H(\sigma)\) for \(\sigma = \text{cdh}(A,Y)\text{cdh}(X,B)\text{cdh}(X,Y)\text{cdh}(A,B)\hat{A}\hat{B}\).

Informally, the security notion expected of such protocols is the following. If \(\hat{A}\) completes a session with (honest) peer \(\hat{B}\), then the session string computed by \(\hat{A}\) is indistinguishable from a random bitstring for everyone except \(\hat{B}\). It has been shown by Canetti and Krawczyk [42] that this is sufficient to establish a secure channel between \(\hat{A}\) and \(\hat{B}\). The secure channel can then be used for key confirmation. Recent adversary models like eCK build on this definition, but also allow for many scenarios where the adversary learns additional information such as ephemeral secret keys, static secret keys, or session keys. The eCK-model guarantees resilience against Unknown Key Share Attacks, Key Compromise Impersonation Resilience, and Weak Perfect Forward Secrecy, which we discuss at the end of Section 3.2.1.

3.1.3 EasyCrypt

EasyCrypt [13, 10] is a machine-checked framework for building and verifying security proofs of cryptographic constructions. EasyCrypt follows the code-based, game-based approach to reductionist arguments: a proof consists of a series of probabilistic programs with adversarial code, called games, and of probabilistic claims relating the probability of one or more events in one or more games. However, EasyCrypt adopts a foundational approach, meaning that probabilistic claims, and the overall security statement, must all be justified to the last detail by means of elementary rules. Leveraging the state of the art in program verification, all probabilistic claims are proved using a probabilistic Relational Hoare Logic (pRHL), which generalizes Relational Hoare Logic [32] to a probabilistic setting. pRHL is a program logic whose judgments are of the form

\[
\models \Phi \sim c_1 : c_2 \Rightarrow \Psi
\]

where \(c_1\) and \(c_2\) are games and \(\Phi\) and \(\Psi\) are relations on program states. The rules of pRHL allow to derive valid judgments, where a judgment as above is valid if for every initial memories \(m_1\) and \(m_2\) that are related by \(\Phi\), the output sub-distributions obtained by executing \(c_1\) on \(m_1\) and \(c_2\) on \(m_2\) respectively are related by \(\Psi^\#\), where \(\#\) is an operator.
Chapter 3. Modular Proofs of Key Exchange Protocols

that lifts relations on states to relations on sub-distributions over states. The definition of \# is inspired from probabilistic process algebra. For suitable choices of \( \Psi \), this implies inequalities of the form

\[
\Pr[c_1, m_1 : E_1] \leq \Pr[c_2, m_2 : E_2]
\]

which are typical in game-based proofs, i.e., the probability of event \( E_1 \) after executing \( c_1 \) in initial memory \( m_1 \) is upper-bounded by the probability of event \( E_2 \) after executing \( c_2 \) in \( m_2 \).

Although pRHL captures common patterns of reasoning in cryptographic proofs, there is an impedance mismatch between cryptographic practice and proofs built using pRHL; in particular, pRHL lacks mechanisms to instantiate previous results, and to apply high-level principles in proofs. To make matters precise, consider for instance the reduction of SCDH to CDH: using pRHL, one can prove that any instance of SCDH can be reduced to CDH, but one cannot perform the proof once and for all, and reuse the result. Fortunately, EasyCrypt now features a module system; the module system combines the power of module systems, as they exist in functional programming languages, with a system of capabilities that is used to restrict access to oracles or fragments of memories, as required in cryptography. The module system can be used for performing successive reductions locally, as often featured in pen-and-paper proofs. Module systems are essential to formalize complex proofs such as the ones considered here; indeed, previous attempts to carry out the generic proof without the module system were unsuccessful, because the adversary was carried explicitly throughout the proof, making reasoning unwieldy.

Additionally, the module system allows to prove general principles once and for all, and to carry out proofs simply by applying high-level principles. In our formalization underlying this paper, we make extensive use of the following principles:

- lazy and eager sampling: this is used to switch back and forth between an implementation of a random function in which images are sampled on demand (lazily), or during initialization of the game (eagerly);
- plug and pray: if some event \( \Phi \) happens for some \( 0 \leq i < n \), randomly sample a value \( j \) in this range and consider the event \( \Phi \land i = j \) instead of \( \Phi \); and
- adversary prescience: this is used to provide an upper bound to the probability that an adversary guesses an unused value in the range of a random function.

3.1.4 Trapdoor Test, Twin DH, and (S)CDH

To minimize the EasyCrypt proof effort, we first prove a generalized version of the Twin DH Assumption from [43]. We use this result for the protocol proofs and to obtain a tighter reduction from CDH to SCDH based on Shoup’s self corrector [93].

* Twin DH

In the original Twin DH assumption, the adversary is given challenges \( X, Y, Z \in G \) and has to compute the group elements \( (\text{cdh}(X, Z), \text{cdh}(Y, Z)) \) given oracle access to

\[
2\text{DDH}(\hat{Z}, U, V) \doteq (\text{ddh}(X, \hat{Z}, U) \land \text{ddh}(Y, \hat{Z}, V)).
\]

58
3.1. Background

Game $G_{2DDH}$:

- $x \leftarrow \mathbb{F}_p^2$; $X \leftarrow g^x$
- $y \leftarrow \mathbb{F}_p^2$; $Y \leftarrow g^y$
- $z \leftarrow \mathbb{F}_p^2$; $Z \leftarrow g^z$
- $t \leftarrow A_{2DDH}(X, Y, Z)$
- return $\phi(X, Z, t)$

Game $G$:

- $x \leftarrow \mathbb{F}_p^2$; $X \leftarrow g^x$
- $z \leftarrow \mathbb{F}_p^2$; $Z \leftarrow g^z$
- $t \leftarrow B(X, Z)$
- return $\phi(X, Z, t)$

Figure 3.2: Twin DDH games $G_{2DDH}$ and $G$.

The value $Y$ is called the “twin” of $X$ and the assumption can be seen as a “twin version” of the Strong DH assumption, which is a variant of Gap CDH where the first input of the DDH oracle is fixed. In contrast to these two assumptions, Twin DH follows from CDH in all groups since the $2DDH$ oracle can be simulated using the trapdoor test.

Our generalization uses $n$ twins $Y_1, \ldots, Y_n$ of $X$ instead of a single twin and consequently provides a $2DDH$ oracle that can be used with all twins $X, Y_i$. Concretely, for a predicate $\phi$, we define the two games $G_{2DDH}$ and $G$ given in Figure 3.2 and prove the following Lemma.

**Lemma 3.1.** Let $A$ be a $G_{2DDH}$ adversary that performs at most $q$ queries to $2DDH$. Then there exists a $G$ adversary $B$ such that

$$\Pr [G_{2DDH}(A) = \text{true}] \leq \Pr [G(B) = \text{true}] + q/p.$$

Moreover, $B$ runs in time $O(T_A + q t_G + n t_G)$ where $t_G$ denotes the time required to perform a group operation such as exponentiation or division.

**Proof.** We define $B$ as

$$B(Z, Y) \doteq$$

- $\vec{r} \leftarrow \mathbb{F}_p^n$; $\vec{s} \leftarrow \mathbb{F}_p^n$; $Y_1 \leftarrow g^{s_1} / X^{r_1}$; $\ldots$; $Y_n \leftarrow g^{s_n} / X^{r_n}$
- $t \leftarrow A_{2DDH}(X, \vec{Y}, Z)$
- return $t$

and note that the distribution on $(X, \vec{Y}, Z)$ is the same as in $G_{2DDH}$. To simulate the $2DDH$ oracle, $B$ uses the test $U^{s_i} V = Z^{s_i}$ instead of $\text{ddh}(X, Z, U) \wedge \text{ddh}(Y_i, Z, V)$. The probability that these tests do not agree is at most $1/p$. Since the adversary can perform $q$ queries to $2DDH$, the probability of distinguishing the simulator is at most $q/p$. $\blacksquare$

We define the following games as instantiations of this lemma:

- $\text{CDH}_{2DDH}$ to CDH for $\phi(X, Z, U) \doteq \text{cdh}(X, Z) = U$. 

59
Chapter 3. Modular Proofs of Key Exchange Protocols

- DLOG\textsubscript{2DDH} to DLOG for $\phi(X, Z, x') \doteq X = g^{x'}$.
- SCDH\textsubscript{2DDH} to SCDH for $\phi(X, Z, S) \doteq \text{cdh}(X, Z) \in S$.

* An efficient reduction from SCDH to CDH We have formalized the proof following the approach outlined in [43]. Note that our proof requires recent extension to EasyCrypt for bounding probabilities of events. In previous versions, it was not possible to relate the probability that an adversary who is called twice wins both times to the probability for a single win.

**Theorem 3.2.** Let $A$ be an SCDH adversary that returns a set of size at most $m$. Then there exists a CDH adversary $B$ such that

$$\Pr[\text{SCDH}(A) = \text{true}] \leq \sqrt{\Pr[\text{CDH}(B) = \text{true}]} + \frac{m^2}{q}.$$  

Furthermore, the adversary $B$ runs in time $O(T_A + m^2 t_G)$.

**Proof.** We first prove that

$$\Pr[\text{SCDH}(A) = \text{true}] = \sqrt{\Pr[\text{CDH}\text{\textsubscript{2DDH}}(B) = \text{true}]}$$

where $n = 1$ for CDH\textsubscript{2DDH}, i.e., there is only one twin. To achieve this, we define:

$$B(X, Y, Z) \doteq \begin{array}{l}
u \leftarrow \mathbb{F}_p^* \\
S_1 \leftarrow A(X, Z) \\
S_2 \leftarrow A(Y, Z^u) \\
\text{foreach } (Z_1, Z_2) \in S_1 \times S_2 : \\
\quad \text{if 2DDH}(Z, Z_1, Z_2^{1/u}) \text{ then return } Z_1
\end{array}$$

Since $B$ wins whenever $A$ wins both times, $B$’s winning probability is equal to the square of $A$’s winning probability. We then conclude the proof by applying Lemma 3.1 and observing that the given simulator calls the 2DDH oracle at most $m^2$ times.  

3.2 Model and Generic Proof

In this section, we first introduce our generic protocol model and our versions of the eCK model with and without adversarial key registration. Afterwards, we present our generic proof for protocols that employ the Naxos trick.

3.2.1 eCK\textsubscript{kr} Security and eCK\textsubscript{nkr} Security

We assume given a set Agent of agent identities. We also define the set $\text{Role} = \{I, R\}$ and the function $\cdot^* : \text{Role} \rightarrow \text{Role}$ such that $\text{I}^* = \text{R}$ and $\text{R}^* = \text{I}$.  

60
3.2. Model and Generic Proof

A protocol definition consists of instantiations for the types and functions given in the first column of Table 3.2. These types and functions are instantiated as follows:

- The sequence $H_1 : \mathbb{I}_1 \to \mathbb{O}_1, \ldots, H_k : \mathbb{I}_k \to \mathbb{O}_k$ defines the types of hash functions used by the protocol.
- $Sk$ defines the type of static secret keys, $Pk$ defines the type of static public keys, $Esk$ defines the type of ephemeral secret keys, $Epk$ defines the type of ephemeral public keys, and $Key$ defines the type of session keys.
- The function $Pk$ defines how the static public key is computed from the static secret key and the function $Epk$ defines how the ephemeral public key is computed from the ephemeral secret key and the static secret key.
- The functions $KeyI$ and $KeyR$ define how the session key is computed from the actor's secret data, the peer's public data, and the participants' identities. We use partial functions to capture failure, e.g., if a subgroup element check fails for one of the arguments.

We keep the distributions according to which the static and ephemeral secret keys are sampled implicit and assume they are uniformly sampled unless otherwise stated. The functions $Epk$, $KeyI$, and $KeyR$ can use the hash functions $H_i$. See Table 3.2 for the $\text{Naxos}^{\text{core}}$ instantiation of the generic model. In the next section, we will demonstrate how $\text{Naxos}^{\text{core}}$ can be transformed into $\text{Naxos}$.
3.2.1.2 Protocol Transformations.

We define two transformations $T_{hsk}$ and $T_{nt}$. The first transformation $T_{hsk}$ modifies a protocol to hash the session key. The second transformation $T_{nt}$ modifies a protocol to utilize the Naxos trick. Table 3.2 demonstrates how the Naxos protocol can be obtained by applying the two transformations to Naxos$^{core}$. We assume $T_{hsk}$ is implicitly parameterized by a positive integer $l$ defining the size of the hash function output.

Given a protocol $\Pi$ using hash functions $\vec{H}$, defining types $Sk$, $Pk$, $Esk$, $Epk$, $Key$, and defining functions $Pk$, $Epk$, $KeyI$, $KeyR$, the transformed protocols $T_{hsk}(\Pi)$ and $T_{nt}(\Pi)$ are defined as follows. We obtain $T_{hsk}(\Pi)$ from $\Pi$ by adding a hash function $H: Key \to \{0, 1\}^l$ to $\vec{H}$, changing the type $Key$ to $\{0, 1\}^l$, redefining $KeyI$ in terms of the original $KeyI$ as $ki \mapsto H(KeyI(ki))$, and redefining $KeyR$ analogously. We obtain $T_{nt}(\Pi)$ from $\Pi$ by adding a hash function $h: Esk \times Sk \to Esk$ to $\vec{H}$ and redefining $Epk$ in terms of the original $Epk$ as $(x,a) \mapsto Epk(h(x,a), a)$. We also redefine the key computation to use $h(x,a)$ instead of $x$. Note that the original $Epk$ usually ignores its second input and $a$ is therefore only used as input to $h$ in the computation of the ephemeral public key. We denote the composition of $T_{nt}$ and $T_{hsk}$ with $T_{nt,hsk}$.

3.2.1.3 Security Experiments.

To define the games $eCK_{kr}$ and $eCK_{nkr}$ (with and without adversarial key registration), we first define the type $PSt$ for the state of protocol sessions and the type $Ev$ for the events required to express the security definition. We define $PSt$ as $Role \times Esk \times Epk \times Agent \times Agent \times Epk \perp \times Key \perp$. We define $Ev$ as the data type generated by the constructors

- $EphRev : Epk \to Ev$,
- $KeyRev : Epk \to Ev$,
- $StaticRev : Agent \to Ev$,
- $Accept : Role \times Epk \times Agent \times Epk \to Ev$,
- $Dishonest : Agent \to Ev$.

The main procedure of the games $eCK_{kr,\Pi}(A)$ and $eCK_{nkr,\Pi}(A)$ is given in the first column of Figure 3.3. We assume that the adversary $A$ consists of the two procedures $A_1$ and $A_2$ sharing state. In the games, the adversary is provided with access to the oracles defined in the second column of Figure 3.3 and with random oracle access using wrappers $H^A_1, \ldots, H^A_k$. The oracles $establishHonest$ and $establishDishonest$ (only in $eCK_{kr,\Pi}$) allow the adversary to establish honest agents for which the keys are sampled and dishonest agents where the public key can be chosen. For honest agents, the adversary can control the execution of initiator and responder sessions using $init_1$, $init_2$, and $resp$. Dishonest agents can be used as peers of protocol sessions, but cannot be used as actors since the static secret key is required to execute the protocol. The remaining oracles allow the adversary to reveal static secrets, ephemeral secrets, and session keys.

The adversary wins if he can distinguish the session key of the test session from a random session key and the test session is fresh, i.e., he did not perform forbidden reveal queries. The freshness condition is formalized using the $fresh$ predicate given in Figure 3.4. We use the ephemeral public key to identify a session for session key reveals and ephemeral reveals.
3.2. Model and Generic Proof

<table>
<thead>
<tr>
<th>Game:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{var } \text{evs}: \text{Ev}^* = [ ] )</td>
</tr>
<tr>
<td>( \text{var } \text{sess}: \text{Nat} \rightarrow \text{PSt} = \emptyset )</td>
</tr>
<tr>
<td>( \text{var } \text{sks}: \text{Agent} \rightarrow \text{Sk} = \emptyset )</td>
</tr>
<tr>
<td>( \text{var } \text{pks}: \text{Agent} \rightarrow \text{Pk} = \emptyset )</td>
</tr>
<tr>
<td>( \text{var } i: \text{Nat} = 0 )</td>
</tr>
<tr>
<td>( t \leftarrow \mathcal{A}_1() )</td>
</tr>
<tr>
<td>( (r, x, A, B, Y, k) \leftarrow \text{sess}[t] )</td>
</tr>
<tr>
<td>( b \leftarrow { 0, 1 } )</td>
</tr>
<tr>
<td>( k' \leftarrow \text{Key} )</td>
</tr>
<tr>
<td>( b' \leftarrow \mathcal{A}_2(b' \upharpoonright k') )</td>
</tr>
<tr>
<td>( t \leftarrow (r, X, \hat{A}, \hat{B}, Y) )</td>
</tr>
<tr>
<td>( \text{return } b = b' \land \text{fresh}_{\text{evs}}(t) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Oracles (eCK_{nkr} does not include establishDishonest):</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{init}_1(A, B): \text{Epk} = )</td>
</tr>
<tr>
<td>( i \leftarrow i + 1 )</td>
</tr>
<tr>
<td>( a \leftarrow \text{sks}[\hat{A}] )</td>
</tr>
<tr>
<td>( \text{if } a = \bot \lor A = \hat{B} \text{ then return } \bot )</td>
</tr>
<tr>
<td>( x \leftarrow \text{Esk}; \ X \leftarrow \text{Epk}(x, a) )</td>
</tr>
<tr>
<td>( \text{sess}[i] \leftarrow (\hat{I}, x, \hat{X}, A, \hat{B}, \bot, \bot) )</td>
</tr>
<tr>
<td>( \text{return } (i, X) )</td>
</tr>
<tr>
<td>( \text{resp}(\hat{B}, \hat{A}, X): \text{Epk} = )</td>
</tr>
<tr>
<td>( i \leftarrow i + 1 )</td>
</tr>
<tr>
<td>( \text{if } \hat{A} = \hat{B} \text{ then return } \bot )</td>
</tr>
<tr>
<td>( b \leftarrow \text{sks}[\hat{A}]; \ A \leftarrow \text{pks}[\hat{A}] )</td>
</tr>
<tr>
<td>( y \leftarrow \text{Esk}; \ Y \leftarrow \text{Epk}(y, b) )</td>
</tr>
<tr>
<td>( k \leftarrow \text{KeyR}(y, b, X, A, \hat{B}, \hat{A}) )</td>
</tr>
<tr>
<td>( \text{if } k = \bot \text{ then return } \bot )</td>
</tr>
<tr>
<td>( \text{sess}[i] \leftarrow (\hat{R}, x, \hat{Y}, \hat{B}, \hat{A}, X, k) )</td>
</tr>
<tr>
<td>( \text{evs} \leftarrow \text{accept}(\hat{R}, X, \hat{B}, \hat{A}, X) )</td>
</tr>
<tr>
<td>( \text{return } (i, Y) )</td>
</tr>
<tr>
<td>( \text{establishDishonest}(\hat{A}, A) = )</td>
</tr>
<tr>
<td>( \text{if } \text{pks}[\hat{A}] \neq \bot \text{ then return } \bot )</td>
</tr>
<tr>
<td>( \text{pks}[\hat{A}] \leftarrow A )</td>
</tr>
<tr>
<td>( \text{evs} \leftarrow \text{accept}(\text{Dishonest}(\hat{A})) )</td>
</tr>
<tr>
<td>( \text{init}_2(i, Y) = )</td>
</tr>
<tr>
<td>( (\hat{I}, x, X, \hat{A}, \hat{B}, \hat{Y}, \bot) \leftarrow \text{sess}[i] )</td>
</tr>
<tr>
<td>( \text{if } \hat{Y} \neq \bot \text{ then return } \bot )</td>
</tr>
<tr>
<td>( a \leftarrow \text{sks}[\hat{A}]; \ B \leftarrow \text{pks}[\hat{B}] )</td>
</tr>
<tr>
<td>( k \leftarrow \text{KeyI}(x, a, Y, B, \hat{A}, \hat{B}) )</td>
</tr>
<tr>
<td>( \text{sess}[i] \leftarrow (\hat{I}, x, X, \hat{A}, \hat{B}, Y, k) )</td>
</tr>
<tr>
<td>( \text{if } k = \bot \text{ then return } \bot )</td>
</tr>
<tr>
<td>( \text{evs} \leftarrow \text{accept}(\hat{I}, X, \hat{A}, \hat{B}, Y) )</td>
</tr>
</tbody>
</table>

Figure 3.3: Games eCK_{kr}(\mathcal{A}) and eCK_{nkr}(\mathcal{A}) for \( \mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \) and protocol \( \Pi \).

3.2.1.4 Discussion.

In eCK_{kr}, we allow the actor of the test session to execute sessions with dishonest users, but the actor and peer of the test session itself must be honest. In both eCK_{kr} and eCK_{nkr}, we disallow \( \hat{A} = \hat{B} \) because many deployed protocols disallow this case or use distinct keys.
Chapter 3. Modular Proofs of Key Exchange Protocols

Figure 3.4: Freshness condition for a trace $evs$ and a test session $t = (r, X, \hat{A}, \hat{B}, Y)$.

for different roles. It would be possible to lift this limitation at the cost of additional proof obligations for users of the generic proof.

The freshness condition captures Unknown Key Share Attacks because if $\hat{A}$ establishes a key with $\hat{B}$, but $\hat{B}$ believes that he shares this key with $\hat{C} \neq \hat{A}$, then there are two non-matching sessions with the same session key and one of them can be revealed. It captures Key Compromise Impersonation because leakage of the actors static secret key is allowed for the test session. It also captures Weak Perfect Forward Secrecy because for all sessions where the adversary is passive (there is a matching session), reveals for all ephemeral secrets, except for those of the test session and its matching session, and for all static secrets are allowed. The stronger notion of Perfect Forward Secrecy requires changes to the freshness condition and we leave such an extension of our results open for future work.

In our definition of Naxos, we use the type $\mathcal{G}$ for ephemeral and static public keys. This models an implementation ensuring that these values are elements of $\mathcal{G}$. It is also possible to use a “larger type” and explicitly model the required checks using failure in the key computation functions.

3.2.2 Generic Proof

Before presenting our generic proof, we define three properties of core protocols:

$\mathbf{(P1)}$ The functions $\text{Pk}$ and $\text{Epk}$ are injective.

$\mathbf{(P2)}$ $\text{KeyI}(x, a, Y, B, \hat{A}, \hat{B})$ is efficiently computable from $\text{KeyR}(x, a, Y, B, \hat{A}, \hat{B})$.

$\mathbf{(P3)}$ If two distinct sessions $(X, Y, \hat{A}, \hat{B}, r)$ and $(X', Y', \hat{A}', \hat{B}', r')$ compute the same session key, then $(X, Y, \hat{A}, \hat{B}) = (Y', X', \hat{B}', \hat{A}')$ and $\{r, r'\} = \{I, R\}$.
3.2. Model and Generic Proof

We assume the second property for simplicity. For core protocols, which we consider here, it usually suffices to reorder the key string elements to obtain KeyI(ki) from KeyR(ki). The third property is called strong partnering in [73] and ensures key independence.

3.2.2.1 Exploiting the Naxos Technique.

We exploit that for protocols $T^\text{nt}(\Pi)$ using the Naxos technique, both $x$ and $a$ are required to learn the secret input $h(x, a)$ of $Epk$. This is a consequence of the fact that the value $h(x, a)$ cannot be revealed by the adversary in the eCK model. This decision is motivated by the assumption that honest agents executing the protocol never store the value $h(x, a)$. We can therefore prove security of $\Pi$ in a restricted eCK$^\text{nt}$ game to obtain eCK$^\text{nt}$-security of $T^\text{nt}(\Pi)$. For $m \in \{\text{kr, nkr}\}$, we obtain eCK$^\text{nt}$ from eCK$^\text{m}$ by replacing $ephRev$ with $ephRev^\text{nt}$ as defined below:

$$ephRev^\text{nt}(i, a) : \text{Esk} =$$

$$\text{if } a \neq \text{sks}[\overline{A}, \ldots, \ldots] \text{ then return } \perp$$

$$\text{evs} \leftarrow ^\text{nt} \text{EphRev}(X)$$

$$\text{return } x$$

Informally, our reduction exploits that $x$ for $\Pi$ in eCK$^\text{nt}$ corresponds to $h(x, a)$ for $T^\text{nt}(\Pi)$ in eCK and a query $ephRev^\text{nt}(i, a)$ in eCK$^\text{nt}$ corresponds to the sequence of queries $x \leftarrow ephRev(i)$; $h^A(x, a)$ in eCK.

To state our lemma, we define $\mathcal{A}$ to be a $(q_{se}, q_{ag}, q_{H})$ eCK$^\text{m}$ (or eCK$^\text{nt}_m$) adversary if $\mathcal{A}$ activates at most $q_{se}$ sessions involving at most $q_{ag}$ agents and performs at most $q_{H}$ queries to the random oracle $H^A$.

We use $q_{h}$ to denote the number of queries to the random oracle $h^A$ introduced by the $T^\text{nt}$ transformation.

**Lemma 3.3.** Let $m \in \{\text{kr, nkr}\}$, $\Pi$ be a protocol, and $\mathcal{A}$ a $(q_{se}, q_{ag}, q_{H})$ eCK$^\text{m}$ adversary. Then there is a $(q_{se}, q_{ag}, q_{H})$ eCK$^\text{nt}_m$ adversary $\mathcal{B}$ such that

$$\Pr [eCK^\text{nt}_m, T^\text{nt}(\Pi)(\mathcal{A}) = \text{true}] \leq \Pr [eCK^\text{nt}_m, \Pi(\mathcal{B}) = \text{true}] + \epsilon^\text{nt}$$

where $\epsilon^\text{nt} = 2q_{h}q_{se}/|\text{Esk}| + q_{h}^2/2|\text{Esk}|$. Furthermore, the adversary $\mathcal{B}$ runs in time at most $O(q_{h}t_{pk} + t_{A})$ where $t_{pk}$ is the time required to compute $P_{k}$.

In our EasyCrypt formalization, we explicitly construct the simulator $S$ sketched in the proof below and prove the probability statement for $\mathcal{B} = S(\mathcal{A})$.

**Sketch.** After boundaking the probability of collisions for ephemeral secrets and boundaking the probability of the adversary querying $h^A(x, *)$ for an ephemeral secret $x$ before revealing it, we define a simulator $\mathcal{B}$ that calls $\mathcal{A}$ and handles queries as follows: On queries $\text{init}$ and $\text{resp}, \mathcal{B}$ updates a mapping from the session index $i$ of the started session to the public key $A_i$ of $i$'s actor. For queries $ephRev(i)$, $\mathcal{B}$ samples and stores the value $\overline{x_i}$ ensuring that there are no collisions and that answers are consistent, i.e., $\mathcal{B}$ simulates the
Figure 3.5: Games defining $\text{CSK}_{\text{nt}}(\mathcal{B})$ and $\text{CSK}_{\text{nkr},\Pi}(\mathcal{B})$ for $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ with alternative $eqS$-oracle definitions for kr and nkr.

3.2.2.2 Exploiting the Hashing of the Session Key.

The $\text{CSK}_{\text{nt}}$ and $\text{CSK}_{\text{nkr}}$ models for protocols that employ the Naxos technique are defined by the three games given in Figure 3.5. The winning conditions state that the adversary must compute certain keys. They result from case distinctions where we show that the adversary cannot win unless he queries these keys to $\text{ephRev}_{\text{nt}}$ or the random oracle $H$. We first describe the games and then explain how they are used in the reduction.

$G_{1,m}^{\text{hsk,nt}}$: The adversary is given a static public key $A$, a vector $\vec{Z}$ of ephemeral public keys, and a vector $\vec{c}$ of static secret keys. To win, he must return the static secret key $a$ for $A$. He is given access to a decision oracle that returns $\text{true}$ if the given $k$ is the session secret $c$. On query $h^A(z, c)$, the adversary must compute certain keys. They result from case distinctions where we show that the adversary cannot win unless he queries these keys to $\text{ephRev}_{\text{nt}}$ or the random oracle $H$. We first describe the games and then explain how they are used in the reduction.
### 3.2. Model and Generic Proof

key for a session with session data \((z_i, a, Y, C, \hat{A}, \hat{C})\) where \(z_i\) must be an element of \(\hat{z}\), \(a\) is fixed, and \(\hat{A}, \hat{C}\), and \(Y\) can be arbitrary. For \(m = \text{nt}\), \(C\) can be arbitrary. For \(m = \text{nkr}\), \(C\) must be an element of \(\text{Pk}(\hat{c})\) reflecting that keys are honestly generated.

\(G_{2,m}^{\text{hsk,nt}}\): The adversary is given an ephemeral public key \(X\), a static public key \(B\), a vector \(\hat{Z}\) of ephemeral public keys, and a vector \(\hat{c}\) of static secret keys. He chooses a static secret key \(c_i\) from \(\hat{c}\), an arbitrary ephemeral public key \(Y\), and arbitrary agent identities \(\hat{A}\) and \(\hat{B}\). To win, he must return a set \(S\) that contains the session key for \((x, c_i, Y, B, \hat{A}, \hat{B})\). He is provided with access to a decision oracle that returns \(\text{true}\) if the given \(k\) is the session key for a session with session data \((z_j, b, W, C, \hat{B}, \hat{C})\) where \(z_j\) must be an element of \(\hat{z}\), \(b\) is fixed, and \(W, \hat{B}\), and \(\hat{C}\) can be arbitrary. For \(m = \text{kr}\), the static public key \(C\) of the peer can be arbitrary. For \(m = \text{nkr}\), \(C\) must be an element of \(\text{Pk}(\hat{c})\).

\(G_{3,m}^{\text{hsk,nt}}\): The adversary is given ephemeral public keys \(X, Y\) and a vector \(\hat{c}\) of static secret keys. He chooses static secret keys \(c_i, c_j\) in \(\hat{c}\) and arbitrary \(\hat{A}, \hat{B}\). To win, he must return a set \(S\) that contains the session key for \((x, c_i, Y, \text{Pk}(c_j), \hat{A}, \hat{B})\).

In the reduction, we use \(G_{1,m}^{\text{hsk,nt}}\) to handle the case where the adversary queries \(\text{ephRev}_{nt}(i, a)\) without revealing the static secret \(a\) for some \(\hat{A}\). For the remaining cases, we know that the ephemeral secret \(x\) of the test session must be secret. We use \(G_{2,m}^{\text{hsk,nt}}\) to handle the case where the static secret \(b\) of the test session’s peer remains unrevealed and \(G_{3,m}^{\text{hsk,nt}}\) to handle the case where \(b\) is revealed and there is a matching session with unrevealed ephemeral secret \(y\). The \(\text{eqS}\) oracle in \(G_{1,m}^{\text{hsk,nt}}\) is used to synchronize queries to \(H^{\hat{A}}\) and \(\text{keyRev}\) for \(\hat{A}\)'s sessions. Analogously, \(\text{eqS}\) in \(G_{2,m}^{\text{hsk,nt}}\) is used for \(\hat{B}\)'s sessions. We can now state our main theorem using \(q_e\) (resp. \(q_H\)) to denote the number of queries to the oracle introduced by \(\mathcal{T}^{nt}\) (resp. \(\mathcal{T}^{\text{hsk}}\)).

**Theorem 3.4.** Let \(m \in \{\text{kr}, \text{nkr}\}\) and \(\Pi\) be a protocol satisfying properties \(\text{P1-P3}\). Let \(A\) be a \((q_{se}, q_{ag}, q_H)\) \(\text{eCK}_m\) adversary. Then there are \(\text{CSK}^{nt}_m\) adversaries \(B_1-B_3\) such that

\[
2 \Pr[\text{eCK}_{m, \mathcal{T}^{nt, \text{hsk}}}([\Pi](A) = \text{true})] - 1 \\
\leq \delta_1 \Pr[G_{1,m,II}^{\text{hsk,nt}}(B_1) = \text{true}] + \delta_2 \Pr[G_{2,m,II}^{\text{hsk,nt}}(B_2) = \text{true}] \\
+ \delta_3 \Pr[G_{3,\Pi}^{\text{hsk,nt}}(B_3) = \text{true}] + \epsilon_{\mathcal{T}^{nt, \text{hsk}}}
\]

for \(\epsilon_{\mathcal{T}^{nt, \text{hsk}}} = \frac{(2q_eq_{se} + 2q_H^2) / |\text{Esk}|}{\text{q}_{\text{e}}}\), \(\delta_1 = q_{ag}\), \(\delta_2 = q_{ag}q_{se}\), and \(\delta_3 = q_{se}^2\). Furthermore, the adversaries \(B_1\) and \(B_2\) perform at most \(q_Hq_{se}\) queries to \(\text{eqS}\) and the adversaries \(B_2\) and \(B_3\) return sets of size at most \(2q_H\). The adversaries \(B_1-B_3\) run in time at most \(\mathcal{O}(q_{\text{h}} + q_{\text{ag}})t_P + q_{\text{se}}t_{\text{proto}} + q_{\text{se}}q_{\text{h}} + t_A\) where \(t_{\text{proto}}\) denotes the time to execute a protocol session.

**Sketch.** We first apply Lemma 3.3. Then it remains to prove that \(\text{CSK}^{nt}_m\)-security of \(\Pi\) implies \(\text{eCK}^{nt}_m\)-security of \(\mathcal{T}^{\text{hsk}}(\Pi)\). Let \(\sigma\) denote the input to \(H\) used to compute the
Chapter 3. Modular Proofs of Key Exchange Protocols

Figure 3.6: Intermediate game $GI$ used in reductions ($eCK_{nt}$ to $CSK_{nt}$ and $eCK$ to $CSK$).

<table>
<thead>
<tr>
<th>Game:</th>
<th>Oracles:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{var evs : Ev}^* = []$</td>
<td>Replace $keyRev$ with $eqS$.</td>
</tr>
<tr>
<td>$\text{var sks : Agent } \rightarrow \text{ Sk} = \emptyset$</td>
<td>Keep other oracles.</td>
</tr>
<tr>
<td>$\text{var sess : Nat } \rightarrow \text{ PSt} = \emptyset$</td>
<td></td>
</tr>
<tr>
<td>$(S, t) \leftarrow A_1()$</td>
<td>$eqS(i, k) : \text{Key} =$</td>
</tr>
<tr>
<td>$(r, _, X, \hat{A}, \hat{B}, Y, k) \leftarrow \text{ sess}[t]$</td>
<td>$(r, _, X, \hat{A}, \hat{B}, Y, k') \leftarrow \text{ sess}[i]$</td>
</tr>
<tr>
<td>return $(k \in S \land \text{ fresh}_\text{evs}(r, X, \hat{A}, \hat{B}, Y))$</td>
<td>$\text{evs} \leftarrow \text{ evs} . \text{KeyRev}(r, X, \hat{A}, \hat{B}, Y)$</td>
</tr>
<tr>
<td>$sk(\hat{A}) =$</td>
<td>return $(k = k')$</td>
</tr>
<tr>
<td>if $\text{sks}[\hat{A}] = \bot$ then $\text{sks}[\hat{A}] \leftarrow \text{ Sk}$</td>
<td></td>
</tr>
<tr>
<td>return $\text{sks}[\hat{A}]$</td>
<td></td>
</tr>
<tr>
<td>$pk(\hat{A}) =$</td>
<td></td>
</tr>
<tr>
<td>return $\text{Pk}(sk(\hat{A}))$</td>
<td></td>
</tr>
</tbody>
</table>

session key of the test session. We first boundake the probability that the adversary wins without querying $\sigma$ to $H_A$ by $1/2$. First, note that he cannot reveal a session key with hash input $\sigma$ since condition $P3$ for $\Pi$ implies that the corresponding session is either a matching session or the test session itself (up to collisions of ephemeral secrets and guessing of unused ephemeral secrets). He therefore receives a key that is sampled independently of his view for both values of $b$ and cannot do better than guessing $b$ in this case.

We now proceed by boundakeing the probability of $\sigma \in Q_H \land \text{ fresh}_\text{evs}(sid)$ in $eCK_{nt,m,T^{\text{rev}}(\Pi)}$ where $\text{sid} = (r, X, \hat{A}, \hat{B}, Y)$ and $Q_H$ is the set of values queried to $H$ by the adversary. Our goal is to perform a reduction to the intermediate game $GI_{m,\Pi}$ shown in Figure 3.6. The simulator will use the $eqS$ oracle in $GI_{m,\Pi}$ to simulate the oracles $H^A$ and $keyRev$ and return $(t, Q_H)$. The $eqS$ oracle is used to synchronize values returned in $keyRev$ and $H$, but it cannot be used for the call to $H$ for $\sigma$ in the main body. We therefore perform a sequence of steps that includes enforcing a (monotonous version of) freshness to remove this call before performing the reduction.

To obtain the three games $G^{\text{hsk}_{nt}}_{1,m,\Pi}$, $G^{\text{hsk}_{nt}}_{2,m,\Pi}$, and $G^{\text{hsk}_{nt}}_{3,m,\Pi}$ from $GI_{m,\Pi}$, we perform two case distinctions followed by one reduction for each case. The first case distinction is for the event that the adversary queries $\text{EphRev}(i, a)$ without performing $\text{StaticRev}(\hat{A})$ and revealing $a$ beforehand. To boundake this probability, we first guess $\hat{A}$ and then perform a reduction to $G^{\text{hsk}_{nt}}_{1,m,\Pi}$. Since the adversary can reveal all secrets except for $a$ and the ephemeral secret keys of $\hat{A}$, the simulator receives the static secret keys $\vec{c}$ of the other agents, the ephemeral public keys of $\hat{A}$’s sessions, and $A$. The simulator samples all other values himself and can simulate all oracles on its own, except for $eqS$ where the provided oracle is used for $\hat{A}$’s sessions. If $m = nk$, all keys are honestly generated and for all queries to $eqS$, the static public key of the peer is equal to $\text{Pk}(c)$ for some $c \in \vec{c}$. If $m = kr$,
3.3. Case Studies

the static public key of the peer can be arbitrary.

Before performing the second case distinction, we guess the test session. Since there is no ephemeral reveal without a previous static reveal, the test session’s ephemeral secret $x$ cannot be revealed. We now perform a case distinction if the adversary reveals the static secret key $b$ of the peer $\hat{B}$ of the test session. If not, then we know that $x$, $b$, and the ephemeral secret keys of $\hat{B}$’s sessions are secret. To perform the reduction to $G^{\text{hsk,nt}}_{2,m,\Pi}$, we guess $\hat{B}$ and define a simulator that receives $X$, $B$, the static secret keys $\vec{c}$ of all agents except $\hat{B}$, and the ephemeral public keys of $\hat{B}$’s sessions. The simulator samples all other values himself and can simulate all oracles on its own, except for $eqS$ where the provided oracle is used for $\hat{B}$’s sessions. Like in the previous case, the static public key of the peer is equal to $Pk(c)$ for some $c \in \vec{c}$ if $m = nkr$ and arbitrary otherwise.

In the last case, there is a static reveal for the peer $\hat{B}$ of the test session. Hence, there must be a matching session with ephemeral secret key $y$ and the only other value that cannot be revealed is $x$. We guess the matching session and since $eqS$ queries for the test session and the matching session are forbidden, the simulator for $G^{\text{hsk,nt}}_{3,\Pi}$ can simulate the $eqS$ oracle on its own in this case.

3.3 Case Studies

We first present our formalization of the Trapdoor Test from [43] and related material. Afterwards, we present our proofs of Nets and Naxos.

3.3.1 Proofs for Naxos and Naxos+

We first prove that Naxos is secure in our eCK_{nkr} model with honestly generated keys under the CDH assumption. In the proof, we discuss why it does not generalize to the eCK_{kr} model with adversarial key registration. Afterwards, we describe two ways to obtain a proof in the eCK_{kr} model from the instantiation of our generic proof. First, the proof can be performed with respect to the Gap-CDH assumption. Second, the protocol can be extended with an additional group element in the key yielding the Naxos+ protocol [79] which was proved secure under the CDH assumption in a model similar to eCK_{kr}.

eCK_{nkr}-security of Naxos under the CDH assumption.

The following theorem states that Naxos is secure in our model without adversarial key registration if the CDH problem is hard.

**Theorem 3.5.** Let $A$ be a $(q_{sc}, q_{ag}, q_{H})$ eCK_{nkr} adversary. Then there exist adversaries $C_1$, $C_2$, and $C_3$ such that

\[
2 \Pr \left[ \text{eCK}_{nkr,\text{Naxos}}(A) = \text{true} \right] - 1 \leq \delta_1 \left( \Pr \left[ \text{DLOG}(C_1) = \text{true} \right] + q_H a_{nkr/p} \right)
\]

69
Chapter 3. Modular Proofs of Key Exchange Protocols

\[ + \delta_2 \left( \sqrt{\Pr[CDH(C_2) = \text{true}] + 4q_H^2/p + q_H q_{se}/p} \right) \]

\[ + \delta_3 \left( \sqrt{\Pr[CDH(C_3) = \text{true}] + 4q_H^2/p} \right) + \epsilon\text{-}\text{DLOG} \]

where \( \delta_1, \delta_2, \delta_3, \) and \( \epsilon\text{-}\text{DLOG} \) are defined as in Theorem 3.4. Furthermore, \( C_1, C_2, \) and \( C_3 \) run in time at most \( O(\log(t_G + t_A)) \) where \( n = \max\{q_h, q_{ag}, q_H q_{se}, q_H^2\} \).

**Proof.** The definition of Naxos\textsuperscript{core} is given in Figure 3.2. It is easy to check that Naxos\textsuperscript{core} satisfies P1–P3 and Naxos = \( \mathcal{T}\text{-}\text{DLOG} \). We can therefore apply Theorem 3.4 to reduce eCK\textsuperscript{nt} security of Naxos to CSK\textsuperscript{nt} security of Naxos\textsuperscript{core}. This step accounts for the loss of eCK\textsuperscript{nt} and yields adversaries \( B_1-B_3 \) that return sets of size at most \( 2q_H \) and perform at most \( q_H q_{se} \) queries to eqS. In the next step, we will define \( C_1, C_2, \) and \( C_3 \) and prove that the inequalities

\[ \Pr[C_{1, \text{nt}}(B_1) = 1] \leq \Pr[\text{DLOG}(C_1) = \text{true}] + q_H q_{se}/p \]

\[ \Pr[C_{2, \text{nt}}(B_2) = 1] \leq \sqrt{\Pr[\text{CDH}(C_2) = \text{true}] + 4q_H^2/p + q_H q_{se}/p} \]

\[ \Pr[C_{3, \text{nt}}(B_3) = 1] \leq \sqrt{\Pr[\text{CDH}(C_3) = \text{true}] + 4q_H^2/p} \]

hold where \( C_{i, \text{nt}} \) denotes the corresponding CSK\textsuperscript{nt} game instantiated with Naxos\textsuperscript{core}.

**Game \( G_{1, \text{nt}} \).** Instantiating with Naxos\textsuperscript{core} yields:

\[ a \leftarrow \mathbb{F}_p; \quad A \leftarrow g^a \]

\[ \bar{z} \leftarrow \mathbb{F}_p^{q_{ag}}; \quad \bar{Z} \leftarrow g^{\bar{z}} \]

\[ \bar{c} \leftarrow \mathbb{F}_p^{q_H-1} \]

\[ a' \leftarrow B_1(A, \bar{Z}, \bar{c}) \]

\[ \text{return } (a = a') \]

\[ \text{eqS}(i, Y, j, \hat{A}, \hat{C}, k) = \]

\[ \text{return } k = (\text{cdh}(A, Y)Z_i^j \text{cdh}(Z_i, Y)\hat{A}\hat{C}) \]

Since we perform a reduction to DLOG\textsuperscript{2DDH} in the first step, we have already rewritten eqS such that it does not use \( a \) and \( \bar{z} \). Before continuing, we rename DLOG\textsuperscript{2DDH} such that \( X \) becomes \( A \), \( \hat{Y} \) becomes \( \bar{Z} \), and \( Z \) becomes \( R \). Our DLOG\textsuperscript{2DDH} adversary \( C_i' \) then receives the DLOG-challenge \( A \), the twins \( \bar{Z} \) of \( A \), and the value \( R \) which is unused in DLOG. \( C_i' \) samples \( \bar{c} \), calls \( B_1 \) with \( (A, \bar{Z}, \bar{c}) \), and returns \( B_1 \)’s return value, which is equal to \( \text{dlog}(A) \) whenever \( B_1 \) wins. \( C_i' \) uses the following implementation to simulate eqS:

\[ \text{eqS}(i, Y, j, \hat{A}, \hat{C}, k) = \]

\[ (U_1U_2U_3\hat{D}E) \leftarrow k \]

\[ d \leftarrow \text{2DDH}(i, U_1, U_3) \]

\[ \text{return } (d \land U_2 = Z_i^j \land \hat{D} = \hat{A} \land \hat{E} = \hat{C}) \]
3.3. Case Studies

Since the original $eqS$ returns true if and only if $U_1 = \text{cdh}(A, Y)$ and $U_3 = \text{cdh}(Z_i, Y)$ (which corresponds to the 2DDH result) and the remaining equalities hold, the simulation is perfect. We can now apply Lemma 3.1 to obtain a reduction to DLOG for an adversary $C_1$.

While this reasoning step is valid in the eCKkr model, it does not work in the eCKkr model since the adversary can register arbitrary static public keys. Hence, the $eqS$ oracle takes $C \in \text{Pk}$ instead of an index $j$ into $\vec{c}$. In this case, we cannot check if $U_2 = \text{cdh}(Z_i, C)$ by performing the test $U_2 = Z^c_j$ in the implementation of $eqS$ for the simulator.

Game $G_{n,nt}^{\text{shk,nt}}$. Instantiating with Naxos$^{\text{core}}$ yields:

$\begin{align*}
x &\xleftarrow{\$} \mathbb{F}_p; & X &\leftarrow g^x \\
b &\xleftarrow{\$} \mathbb{F}_p; & B &\leftarrow g^b \\
\vec{z} &\xleftarrow{\$} \mathbb{F}_p^{q-1}; & \vec{Z} &\leftarrow g^{\vec{z}} \\
\vec{c} &\xleftarrow{\$} \mathbb{F}_p^{q-1} \\
(i, Y, \hat{A}, \hat{B}, S) &\leftarrow B_2(X, B, \vec{c}, \vec{Z}) \\
\text{return } (Y^\nu \text{cdh}(B, X) \text{cdh}(X, Y) \hat{A}\hat{B}) \in S  \\
eq S(j, W, u, \hat{B}, \hat{C}, k) = \\
\text{return } k = (\text{cdh}(B, W) Z^\nu_j \text{cdh}(Z_j, W) \hat{B}\hat{C})
\end{align*}$

For this game, we perform the reduction in three steps. The first reduction is to SCDH$^{2\text{DDH}}$ for which we define the adversary $C'_2$. Then we use Lemma 3.1 to get rid of the 2DDH oracle and finally Theorem 3.2 to transform the SCDH adversary into a CDH adversary which yields the adversary $B_2$. Before continuing, we rename SCDH$^{2\text{DDH}}$ such that $X$ becomes $B$, $\vec{Y}$ becomes $\vec{Z}$, and $Y$ becomes $X$. The CDH challenge is therefore $B, X$ and $\vec{Z}$ is the vector of twins of $B$ for which the 2DDH oracle can be used.

We define the SCDH$^{2\text{DDH}}$ adversary $C'_2$ as follows. $C'_2$ gets $B, \vec{Z}, X$ as input, samples $\vec{c}$, calls $B_2$ with these values, and gets $(i, Y, \hat{A}, \hat{B}, S)$. To transform $S$ into a set that contains $\text{cdh}(B, X)$ whenever $B_2$ wins, $C'_2$ applies the function $(U_1 U_2 U_3 \hat{A}\hat{B}) \mapsto U_2$. To (perfectly) simulate the orginal $eqS$, $C'_2$ uses the implementation

$\begin{align*}
\text{eqS}(j, W, u, \hat{B}, \hat{C}, k) = \\
(U_1 U_2 U_3 \hat{D}\hat{E}) &\leftarrow k \\
d &\leftarrow 2\text{DDH}(i, U_1, U_3) \\
\text{return } (d \land U_2 = Z^\nu_j \land \hat{D} = \hat{B} \land \hat{E} = \hat{C}).
\end{align*}$

Since the adversary can register arbitrary static public keys in the eCKkr model, the $eqS$ oracle in the kr version of this game takes $C \in \text{Pk}$ instead of an index $u$ into $\vec{c}$. In this case, we cannot check if $U_2 = \text{cdh}(Z_j, C)$ by performing the test $U_2 = Z^\nu_j$ in the implementation of $eqS$ for the simulator.
Chapter 3. Modular Proofs of Key Exchange Protocols

Game $G_{3,nkr}^{hsk,nt}$. Instantiating with Naxos$^{core}$ yields:

$$
\begin{align*}
&x \xleftarrow{\$} \mathbb{F}_p; \quad X \xleftarrow{\$} g^x \\
y \xleftarrow{\$} \mathbb{F}_p; \quad Y \xleftarrow{\$} g^y \\
\tilde{c} \xleftarrow{\$} Sk^{hsk} \\
(i, j, \hat{A}, \hat{B}, S) \leftarrow B_3(X, Y, \tilde{c}) \\
\text{return } (Y^c X^\tilde{c}) \text{cdh}(X, Y) \hat{A} \hat{B}) \in S
\end{align*}
$$

We can directly perform a reduction to SCDH and then use Theorem 3.2 to obtain a reduction to CDH. For the reduction to SCDH, we use the function $(U_1 U_2 U_3 \hat{D} \hat{E}) \mapsto U_3$ to transform $S$ into a set that contains cdh$(X, Y)$. This case directly generalizes to $eCK_{kr}$ since the third game is identical in this case.

$eCK_{kr}$-security of Naxos and Naxos$^{+}$.

In the previous proof, we have pointed out where the proof breaks down in the $eCK_{kr}$ model. We will now describe how to adapt the proof to (1) prove $eCK_{kr}$-security of Naxos under the Gap-CDH assumption and (2) prove $eCK_{kr}$-security of Naxos$^{+}$ under the CDH assumption.

For the proof with respect to Gap-CDH, we can deal with all the problematic cases by calling the DDH oracle with the right input, e.g., with DDH$(Z_i, C, U_2)$ for the first game. Note that there is no need for the twinning technique at all in this case and our generic proof can be instantiated in a very similar way to the original Naxos proof.

The Naxos$^{+}$-core protocol can be obtained from the Naxos$^{core}$ protocol by adding the additional group element cdh$(A, B)$ to the key. Concretely, we define

$$
\begin{align*}
\text{Key} &= G^4 \times \text{Agent}^2, \\
\text{Key}_I(x, a, Y, B, \hat{A}, \hat{B}) &= Y^a B^x A^b \hat{A} \hat{B}, \text{ and} \\
\text{Key}_R(y, b, X, A, \hat{B}, \hat{A}) &= A^y X^b Y^a \hat{A} \hat{B}.
\end{align*}
$$

The additional group element is only required to simulate the eqS oracle. Everything else, in particular the case Game $G_{3,nkr}^{hsk,nt}$, can be trivially adapted.

Game $G_{1,kr}^{hsk,nt}$. For Naxos$^{+}$, we must simulate the following eqS oracle (we underline the differences to the Naxos version):

$$
\begin{align*}
\text{eqS}(i, Y, C, \hat{A}, \hat{C}, k) = \\
\text{return } k = (\text{cdh}(A, Y) \text{cdh}(Z_i, C) \text{cdh}(Z_i, Y) \text{cdh}(A, C)) \hat{A} \hat{C}
\end{align*}
$$

By using the 2DDH oracle for the group elements 1&3 and 2&4, we obtain the following implementation of eqS:

$$
\begin{align*}
\text{eqS}(i, Y, j, \hat{A}, \hat{C}, k) = \\
(U_1 U_2 U_3 U_4 \hat{D} \hat{E}) \leftarrow k \\
d_1 \leftarrow 2\text{DDH}(i, U_1, U_3) \\
d_2 \leftarrow 2\text{DDH}(i, U_2, U_4) \\
\text{return } (d_1 \land d_2 \land \hat{D} = \hat{A} \land \hat{E} = \hat{C})
\end{align*}
$$
3.4. Protocols Without Naxos Trick

The simulation is perfect because the 2DDH calls returns true if and only if

\[ U_1 = \text{cdh}(A, Y) \land U_3 = \text{cdh}(Z_i, Y) \] and

\[ U_4 = \text{cdh}(A, C) \land U_2 = \text{cdh}(Z_i, C). \]

**Game** \( G_{2,kr}^{\text{hsk,nr}} \). For Naxos+, we must simulate the following \( eqS \) oracle

\[
eq S(j, W, C, B, C, k) = \text{return } k = (\text{cdh}(B, W) \text{cdh}(C, Z_j) \text{cdh}(Z_j, W) \text{cdh}(B, C) \hat{B} \hat{C})
\]

By using the 2DDH oracle first for the group elements 1 and 3 and then using the oracle in a second call for the group elements 2 and 4, we obtain the following implementation of \( eqS \).

\[
eq S(i, Y, j, A, C, k) = \]

\[
(U_1 U_2 U_3 U_4 \hat{D} \hat{E}) \leftarrow k \\
 d_1 \leftarrow 2DDH(i, U_1, U_3) \\
 d_2 \leftarrow 2DDH(i, U_4, U_2) \\
 \text{return } (d_1 \land d_2 \land \hat{D} = A \land \hat{E} = C)
\]

The simulation is perfect because the first 2DDH calls returns true iff \( U_1 = \text{cdh}(B, W) \land U_3 = \text{cdh}(Z_j, W) \) and the second call returns true iff \( U_4 = \text{cdh}(B, C) \land U_2 = \text{cdh}(Z_j, C) \).

3.3.2 Proofs for Nets

The proofs for Nets are structured very similarly to the corresponding Naxos proofs and yield similar bounds. We therefore summarize the required changes in this section and refer to our EasyCrypt formalization for details.

The proof that Nets is secure in our eCK\(_{nk} \) model with honestly generated keys under the CDH assumption is follows the corresponding proof for Naxos. The only significant difference is how the 2DDH oracle is used to simulate the \( eqS \) oracles in the first and second games. Whereas the Naxos protocol uses the concatenation of three group elements in the key, Nets uses the concatenation of two group elements \( U_1 U_2 \) where

\[ U_1 = \text{cdh}(A, B) \text{cdh}(A, Y) \text{cdh}(X, B) \text{cdh}(X, Y) \] and

\[ U_2 = \text{cdh}(X, Y) \]. Computing the right queries to 2DDH for simulating \( eqS \) requires divisions. Concretely, the first game uses 2DDH\((U_1/A) z_i^{u_2} z_2^{-u_2}, U_2)\) and the second game uses 2DDH\((U_1/B) z_i^{u_2} z_2^{-u_2}, U_2)\).

To prove eCK\(_{kr} \)-security under the Gap-CDH assumption, it is again possible to closely follow the original proof and use the DDH oracle to simulate \( eqS \).

3.4 Protocols Without Naxos Trick

In this section, we describe our generic proof for protocols that do no utilize the Naxos trick and its application to a version of HMQV. The results of this section have not been formalized in EasyCrypt and we leave this open for future work.
Theorem 3.6. Let \( \Pi \) be a protocol that satisfies P1–P3. For all efficient adversaries that win the eCK_{kr,\Pi} game with non-negligible probability, there exists an efficient adversary that wins one of the CSK_{kr,\Pi} games with non-negligible probability.

The proof is analogous to the proof of Theorem 3.4 and appears in the full version of the paper [8]. The proof performs a different case distinction with respect to the reveal queries performed by the adversary than the proof of Theorem 3.4.

3.4.2 eCK_{kr}-security of mHMQV under the Gap-CDH assumption

We first define our (modified version) mHMQV^{core} as follows. We use \( Pk(a) = g^a \) and \( Epk(x, a) = g^x \) for ephemeral and static key computation. Using the hash function \( h : \)
### 3.4. Protocols Without Naxos Trick

\( \mathbb{G} \rightarrow \mathbb{F}_p \), we define the session keys:

\[
\text{Key}_I(x, a, Y, B, \hat{A}, \hat{B}) = (Y B^{hy})^{x + a h x} \hat{A} \hat{B} XY \\
\text{Key}_R(y, b, X, A, \hat{B}, \hat{A}) = (X A^{hx})^{y + b h y} \hat{A} \hat{B} XY 
\]

We instantiate the types using \( \mathbb{G} \) for group elements and \( \mathbb{F}_p \) for exponents. We then define \( \text{mHMQV} \) as \( \mathcal{T}^{hsk}(\text{mHMQV}^\text{core}) \). A similar version of HMQV has been proposed in the original paper [71, Remark 9.1] for compatibility between the variants with one, two, and three passes. We slightly deviate from the original definition by removing the identities from \( h \)'s input (like in MQV) and including \( A, B, X \) and \( Y \) as input to the key derivation hash. Including additional session data in the hash is considered a prudent engineering principle [40] because it ensures agreement on this data. Second, it allows us to apply our generic proof directly since the resulting protocol satisfies \( \text{P3} \). To make the protocol symmetric, it would be possible to sort the tuples \( A, X \) and \( B, Y \) to determine the order of these elements. We prove the following theorem for \( \text{mHMQV} \).

**Theorem 3.7.** If there is an efficient adversary that wins the \( \text{eCK}^\prime_{\text{mHMQV}} \) game with non-negligible probability, then there is an efficient adversary that wins the Gap-CDH game with non-negligible probability.

**Sketch.** Since \( \text{mHMQV}^\text{core} \) satisfies \( \text{P1-P3} \), we can apply Theorem 3.6 and prove \( \text{CSK}_h \)-security of \( \text{mHMQV}^\text{core} \). As in the \( \text{Nets} \) proof, we ignore the public part \( \hat{A} \hat{B} XY \) in our discussion of winning conditions and \( eqS \). Before discussing the individual games, we note that under the Gap-CDH assumption which provides a DDH-oracle, it is possible to simulate the \( eqS \) oracle in all of the games since at least the secret key \( z_i \) is always known. To simulate \( eqS \) queries, e.g., in \( G_1^{hsk} \), it suffices to compute

\[
W^{z_i} \text{cdh}(C, W)^{h(g^{z_i})} D^{z_i} h(W) \cdot \text{cdh}(C, D)^{h(g^{z_i})} h(W) = k
\]

for \( z_i \) in \( \bar{z} \), \( C \in \{A, B\} \), and \( W, D, k \) arbitrary. To achieve this, the DDH oracle can be used to check

\[
\text{cdh}(C, W)^{h(g^{z_i})} D^{z_i} h(W) = \frac{k}{W^{z_i} E^{z_i} h(W)}.
\]

For game \( G_1^{hsk} \), we perform a reduction to Gap-CDH using the Forking Lemma [90]. We know there exists an adversary \( A \) such that for the CDH challenge \( A, B \) and uniformly sampled \( \bar{z} \), the call \( A(\bar{z}, A, B) \) returns \( i, Y \), and a set \( S \) that contains

\[
Y^{z_i} \text{cdh}(A, Y)^{h(Z_i)} B^{z_i} h(Y) \cdot \text{cdh}(A, B)^{h(Z_i)} h(Y)
\]

with non-negligible probability. To apply the Forking Lemma from [27], we use \( A \) to define a randomized algorithm \( B \) that returns \( v \in [g_1] \), \( \text{cdh}(A, Y)^{v} \text{cdh}(A, B)^{h(Y)} \), and \( Y \) such that \( Y \) is the \( v \)-th query to \( h \) with non-negligible probability. First, \( B \) guesses \( v \), then it calls \( A \) with the CDH challenge \( A, B \) and uniformly sampled \( \bar{z} \). \( B \) then computes the result from \( A \)'s return values \( i, Y, S \) as follows. If \( Y \) is not the \( v \)-th query, \( B \) returns \( \perp \). Otherwise,
$B$ divides all elements of $S$ by $Y^{z_i} B^{z_i \bar{h}(Y)}$, exponentiates the result with $1/\bar{h}(g^{z_i})$, and uses the DDH-oracle to search for $U$ with $\text{cdh}(A, Y B^{\bar{h}(Y)}, U)$. If there is no such value, $B$ returns $\bot$. Otherwise, $B$ returns $v, Y, \text{cdh}(A, Y) \text{cdh}(A, B)^{\bar{h}(Y)}$. The Forking Lemma yields a randomized algorithm $C$ from $B$ that returns $Y, \text{cdh}(A, Y) \text{cdh}(A, B)^e, \text{cdh}(A, Y) \text{cdh}(A, B)^{e'}$ with $e \neq e'$ with non-negligible probability. Intuitively, the algorithm first calls $B$ to obtain $v, Y, \text{cdh}(A, Y) \text{cdh}(A, B)^e$ for $e = \bar{h}(Y)$. Then, it calls $B$ again using the same randomness, but resampling the values returned by $\bar{h}$ for all query-indices greater or equal than $v$, i.e., $e' = \bar{h}(Y)$ is the first value that differs. We can then compute

$$\text{cdh}(A, B) = \left( \frac{\text{cdh}(A, Y) \text{cdh}(A, B)^e}{\text{cdh}(A, Y) \text{cdh}(A, B)^{e'}} \right)^{\frac{1}{1-e'}}.$$ 

For game $G_{2}^{\text{hsk}}$, we also reduce to Gap-CDH. We know there exists an adversary $A$ such that for the CDH challenge $X, B$ and uniformly sampled $\vec{c}$ and $\vec{z}$, the call $A(\vec{c}, \vec{z}, X, B)$ returns $i, Y$, and a set $S$ that contains

$$\text{cdh}(X, Y) Y^{z_i} \bar{h}(X) \text{cdh}(X, B)^{i \bar{h}(Y)} B^{c_j \bar{h}(X) \bar{h}(Y)}$$

with non-negligible probability. Using a similar approach as before, we can obtain an algorithm that returns the group element $\text{cdh}(X, Y) \text{cdh}(X, B)^e$ and the group element $\text{cdh}(X, Y) \text{cdh}(X, B)^{e'}$ for $e \neq e'$ with non-negligible probability. We can then compute $\text{cdh}(X, B)$ like in the previous case.

For $G_{3}^{\text{hsk}}$, the reduction to Gap-CDH is simpler than the previous two cases since we know two secret keys instead of only one. We can call $A$ with randomly sampled $\vec{c}, \vec{z}$, and a CDH challenge $A, Y$. Since $A$ returns $i, j$, and a set $S$ that contains

$$Y^{z_i} \text{cdh}(A, Y)^{i \bar{h}(g^{z_i})} g^{z_i c_j \bar{h}(Y)} A^{c_j h(g^{z_i}) \bar{h}(Y)}$$

with non-negligible probability, we can then use the DDH oracle to find $\text{cdh}(A, Y)$. For $G_{4}^{\text{hsk}}$, we can perform a similar reduction to Gap-CDH for the CDH challenge $X, Y$. ■
Our work on automated analysis of Padding-based Encryption Schemes lies at the intersection between symbolic and computational cryptography, and draws on verification techniques from both areas. We refer to [37, 49] for a recent account of symbolic verification and focus on verification tools and methods for the computational model, and cryptographic soundness.

**Cryptographic soundness**  Cryptographic soundness [2] bridges the gap between the symbolic and computational model by establishing conditions under which constructions shown to be secure in the former, remain secure in the later. This approach was initially applied to the domain of formal encryption but has been extended to many settings [49], but mostly confined to protocols. However, cryptographic soundness for constructions based on exclusive-or and one-way trapdoor permutations has remained elusive. Negative results such as [97] show that cryptographic soundness may indeed be very difficult to achieve for the setting of this dissertation.

**Machine assisted proofs in the computational Model**  CryptoVerif [35] is a tool based on applied $\Pi$--calculus and equivalence that can establish security of many cryptographic protocols in a fully automated way in the computational model. It has been used to verify Full Domain Hash signature schemes, Kerberos [38] and One-Encryption Key Exchange [36].

EasyCrypt [15] supports the construction and verification of cryptographic proofs in the computational model. In addition, EasyCrypt has been applied to verify security of several padding-based schemes, such as OAEP [16].

However, fully automated proofs of padding-based schemes are out of reach of these tools. [50] reports on a Hoare-like logic and an automated tool for proving CPA security of padding-based schemes. The tool can verify the security of several schemes, such as BR [29] and REACT [88], but fails to verify most other examples. In our view, the limitations of the approach are inherited from using a Hoare logic, which favors local reasoning. In
contrast, we use a more global approach in which one reasons about the probability of an event in an experiment. The Hoare-Logic based approach has been adapted to deal with several different kind of constructions, including block cipher modes of operation [60] and almost-universal hash for message authentication codes [59].

In addition, a number of formalisms have been developed to reason about security of cryptographic primitives and constructions in the computational model [66, 16]. However, these systems reason about constructions described in mathematical vernacular, and are thus not readily amenable to automation. Similar formalisms exist for cryptographic protocols [91], but these are not automated either. The Computational Indistinguishability Logic [9] is a proof system that enables reasoning about indistinguishability and negligibility of events. This logic was formalised in Coq [47] and has been used to prove several cryptographic constructions [54, 57].

**Synthesis of Cryptographic constructions** The first application of program synthesis to cryptography is the AVGI toolkit [89]. The toolkit allows users to state security requirements, for instance authentication or secrecy, and non-functional requirements such as message length and available bandwidth. The AVGI toolkit is composed of a protocol generator, that applies pruning techniques to curb the state space explosion problem, and a protocol screener that applies symbolic methods to verify, in the symbolic model, whether generated protocols comply with the desired properties.

Our application of synthesis combined with automated security analysis in the computational model has inspired a stream of work that focuses on different classes of constructions. Malozemoff et al. [81] apply synthesis to construct block-cipher modes of operation satisfying confidentiality. Barthe et al. [12] use a similar approach to synthesize structure preserving signatures building on an extension of the Generic Group Analyzer [11] to interactive assumptions. Hoang et al. [65] build on [81] and extend it to obtain modes of operation that also satisfy integrity. Ambrona et al. [5] build on [12] and develop a new method to automatically prove security statements in the Generic Group Model. In a related line of work, Tiwari et al. [96] define a general purpose program synthesis infrastructure based on SMT solver Yices that allows to express both functional and non-functional application specific requirements. They instantiate their approach to synthesize padding-based encryption schemes and block ciphers modes of operation. In a related line of work, Akinyele et al. [4, 3] have explored the use of transformations for improving efficiency and security of constructions in the setting of pairing-based cryptography.

**Key exchange protocols** There is a vast body of literature on key-exchange protocols and on their associated security models that is directly related to our work on Modular Proofs of Key Exchange Protocols; for a comparison between some existing models we refer to [74, 45, 51].

There has been extensive work on the formal verification of key exchange protocols, see for instance the recent survey [37]. A significant amount of work is carried in the symbolic model.
This level of abstraction makes symbolic model amenable to automated analysis, and many tools have been developed for proving protocol security. Recent results focusing on DH-based key exchange protocols include [23], [75] and [92].

As mentioned earlier, symbolic models traditionally ignore corruption, and thus provide weaker security guarantees than their computational counterpart. Recently, Basin and Cremers [52] have extended symbolic models to account for different corruption models that closely relate to the modeling of corruption in a computational setting. They showed how their models can unveil attacks in protocols that are insecure in the computational setting (with corruption) and yet provably secure in standard symbolic models (without corruption). While their model can be viewed as an intermediate layer between the symbolic and computational models, no soundness results justify their usage. Moreover, some attacks, especially the attack against HMQV reported by Cremers in [52] is based on a stronger form of state reveal, in which the adversary is given not only the ephemeral values, but also computations using them and the long-term key, in contrast to our model, and the discussion in [71]. More recent results in this area include [75] and [92], which focus on unbounded verification of DH-based protocols.
Conclusion

This thesis addresses two challenges in the setting of Cryptography: how to build evidence of the security of crypto-systems as machine-checked proofs in a cost efficient way and how to systematically explore the design space of a given class of construction by using techniques based on program synthesis. The main ingredients used to address this issues are automation and modularity.

Automation   The idea of this work consists in isolating the high level principles used to perform proofs of a class of constructions and capturing them in a logic. It is our hope that these set of high level reasoning principles capture a cryptographer’s intuition in a much closer way than the low level Hoare-logic based reasoning that is used in EasyCrypt proofs.

In addition to that, the limited design space of our core logic makes this domain an ideal target for applying program synthesis. A key ingredient to achieve efficiency and automation is the use of notions like equivalence and deducibility inspired from symbolic cryptography to characterize side conditions and to find attacks. Essentially, our tool-set provides an automated way of finding proofs/attacks for the constructions. This opens an avenue for systematic exploration of the design space and classification of the schemes. While we initially applied this idea to the domain of padding-based encryption schemes, we believe that this high level reasoning principles exist for many different classes of constructions (see Chapter 4).

Moreover, our logic is proved sound in a constructive way, by interpreting judgments in terms of EasyCrypt and establishing the correctness of rules by providing a sketch of proof in the spirit of EasyCrypt proofs. Building a certifying engine that traverses a proof tree from the logic and constructs an EasyCrypt proof is essentially, an engineering task.

The idea of having a high level proof system that uses a more primitive low level proof system as a back-end, is reminiscent to what has been done in programming languages in the last 70 years: you define high level languages and construct mechanisms—compilers—that translate programs in the high level language to a low level language—assembler. In
this setting, we see EasyCrypt as a bedrock that can be used to provide sound and precise meaning to a stack of logics that allow simple and intuitive yet powerful reasoning, without sacrificing the increased trust of machine-checked proofs.

Modularity The crux of this work consists in identifying the common parts of a set of proofs for the same class of constructions and isolate the differences. Then we can re-cast the proofs in terms of the application of a generic proof for the common part and a case-specific proof. We have applied this idea, for the concrete domain of one round authenticated key-exchange protocols. To date, this remains EasyCrypt’s largest case study.

We believe this constitutes a cost-efficient way of doing machine-checked proofs: the cost of doing the generic proof is justified by its multiple applications.

In addition to this, we believe that this work sheds some light on the relation between the security model, in particular, the allowed corruption oracles and the required assumption to complete the proof. Indeed, by stripping the initial model from an intricate freshness condition and a complex oracle system, we end up with a small set of significantly simpler games. At this level of abstraction, it becomes crystal clear which hypothesis is need for each protocol instantiation in each security model.

Building on this simplified model, this work improves the proves by circumventing the need of a decisional Diffie-Hellman oracle and relying on the more standard CDH assumption—although in a model with an admittedly weaker adversary.

Lessons learnt The development of our generic proof for Authenticated Key Exchange protocols constitutes the biggest EasyCrypt development to date. The magnitude of this proof has given valuable insights regarding the overall experience when using the tool.

First, the current implementation has a rich module system, with the particularity that modules are stateful, module quantification is built into the logic and negative restrictions regarding module access can also be stated and checked by the tool. These features are instrumental in supporting modular proofs and instantiation; and proofs by reduction.

Second, judgments in the program logics are “first class citizens” of the base logic. This seemingly unimportant design choice has a great impact: it captures elegantly concepts like logical variables (also called ghost or auxiliary variables) as existentially quantified variables whose scope is includes the pre and post condition. Moreover, this opens the door to the possibility of proving Hoare statements by induction on the value of variables at the beginning of a given piece of code.

Finally, one of the most brilliant design choices of EasyCrypt is the fact that the rules of the program logics can be applied to the programs interactively as tactics and annotations are provided as inputs to the tactics. In the more traditional approach of Hoare logic-based reasoning, source code is directly annotated and then an algorithm traverses the programs and computes verification conditions taking the annotations into account. Afterwards, SMT’s are used to attempt to discharge the proof obligations and the user is presented with the remaining ones. At this point, the relation between the unprovable proof obligations and the annotations and the source code is at best weak. Specially for non-trivial programs
manipulating large state, this verification conditions can be cryptic and the having to debug the specifications in this context leads to an unproductive workflow. To make matters worse, it is often the case that if one wants to attempt a manual proof of the proof obligation in question, one has to resort to an external interactive theorem prover, requiring more domain specific knowledge and adding an additional level of indirection to an already cryptic proof obligation.

The fine-grained control provided by EasyCrypt over how the proof obligations are generated is instrumental and makes the connection between a non-provable proof obligation and the inputted annotations crystal clear. Moreover, one can attempt a manual proof without resorting to an SMT in the same tool, in the same tactic language and in the same logic.

On the down-side, the current implementation of EasyCrypt does not provide good support for defining games in terms of transformations applied other games. This support is fundamental in Cryptography proofs, since they are structured by defining intermediate games as slight modifications of the previous ones. When one of the initial games of a long sequence of games has to be modified, all the intermediate ones have to be modified accordingly. This constitutes a mechanical yet error prone task.

**Future work** A natural direction for future work is to apply automation and synthesis to new classes of constructions. This line of work has already started, as discussed in related work.

One of the most challenging directions is to design a general-purpose logic that can account for arbitrary oracles, challengers and winning conditions; and that can encompass high-level reasoning principles that occur pervasively in proofs such as plug and pray, rejecting queries to oracles under certain conditions, a general reduction rule, etc. One could envision that such a logic can be parametrised by the concrete building blocks used, and by notions of symbolic equivalence and deducibility on these building blocks. Then, the soundness of the logic would hinge on certain properties on these notions. This high-level vernacular of cryptographic proofs can serve as a common ground between cryptography and formal methods practitioners. Moreover, if this is combined with EasyCrypt proof reconstruction, we may be able to further realise Halevi’s program to its most ambitious extent: simpler, high level proofs with complete automation and without giving up on trustworthy end-to-end guarantees.

To conclude, I would like to highlight the really nice interplay between Programming Language/Program Verification topics and Cryptography that we have stumbled upon: challenges coming from the Cryptography community are coped with by using standard techniques from PL/PV. In the process we have built a tool that has a lot of value that transcends the Cryptography setting. I believe these ideas should be studied from a Programming Language and Verification perspective; and exploited to improve the state of the art in Program Verification.
Chapter 5. Conclusion
Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


