Fully Automated Analysis of Padding-Based Encryption in the Computational Model

Gilles Barthe, Juan Manuel Crespo, César Kunz, Benedikt Schmidt[*]

Benjamin Grégoire[†] Yassine Lakhnech[‡] Santiago Zanella-Béguelin[§]

June 2013

Abstract

Computer-aided verification provides effective means of analyzing the security of cryptographic primitives. However, it has remained a challenge to achieve fully automated analyses yielding guarantees that hold against computational (rather than symbolic) attacks. This paper meets this challenge for public-key encryption schemes built from trapdoor permutations and hash functions. Using a novel combination of techniques from computational and symbolic cryptography, we present proof systems for analyzing the chosen-plaintext and chosen-ciphertext security of such schemes in the random oracle model. Building on these proof systems, we develop a toolset that bundles together fully automated proof and attack finding algorithms. We use this toolset to build a comprehensive database of encryption schemes that records attacks against insecure schemes, and proofs with concrete bounds for secure ones.

Keywords: Attack finding, automated proofs, provable security, public-key encryption, static equivalence.

[*]IMDEA Software Institute, Spain.
E-mail: {gilles.barthe,benedikt.schmidt,juanmanuel.crespo,cesar.kunz}@imdea.org
[†]INRIA Sophia Antipolis – Méditerranée, France. E-mail: benjamin.gregoire@inria.fr
[‡]Université de Grenoble, VERIMAG, France. E-mail: yassine.lakhnech@imag.fr
[§]Microsoft Research, UK. E-mail: santiago@microsoft.com

Contents

1 Introduction

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 Goldwasser and Micali [22] 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, which originates from the seminal work of Dolev and Yao [20], 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. This model enables automated analysis, but can miss attacks that are possible in the computational model.

A celebrated article by Abadi and Rogaway [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 [18]. 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 cryptographic constructions directly in the computational model. This alternative is embodied in tools like CryptoVerif [13] and EasyCrypt [5], 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 [3] 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 [12], OAEP+ [30], SAEP [15], or PSS-E [17]. 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, and by evaluating it extensively. The toolset can be used either in batch mode or in interactive mode:

- The batch mode allows to take a census of padding-based encryption schemes. We used it to build a database of more than one million entries. The database 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.
- The interactive mode allows to build proofs or attacks for a selected scheme, and forms the backbone of a cryptographic tutor. The tutor guides users in building security proofs or exhibiting attacks for a scheme chosen by the user or selected from the database. In proof mode, the tutor provides the user with the set of applicable rules at each step of the derivation; upon completion of the derivation, the tutor outputs alternative derivations that deliver better bounds.

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. However, our methods are applicable to other settings; see § 8 for a discussion.

Contents § 2 provides background on public-key encryption schemes and their security; § 3 introduces symbolic tools that are the base of our analyses; § 4 describes logics for chosen-plaintext and chosen-ciphertext security, while § 5 covers attack finding techniques; § 6 reports on the implementation of ZooCrypt and on its evaluation. We conclude with a survey of related work and directions for future work.

Supplemental material, including a web interface for browsing results and a preliminary version of a proof tutor, is available at

`http://easycrypt.info/zoocrypt/`

2 Cryptographic Background

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

2.1 Padding-Based Public-Key Encryption

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

Key Generation The key generation algorithm $\mathcal{KG}$ 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, $\mathcal{E}_{pk}(m)$ outputs a ciphertext c;

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

We require that for pairs of keys (pk, sk) generated by $\mathcal{KG}$, $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$ 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 ℓ 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 family of trapdoor permutations on $\{0,1\}^n$ is a triple of algorithms $(\mathcal{KG}, f, f^{-1})$ such that for any pair of keys (pk, sk) output by $\mathcal{KG}$, f_{pk} and f_{sk}^{-1} are permutations on $\{0,1\}^n$ and inverse of each other. The security of a family of trapdoor permutations is measured by the probability that an adversary (partially) inverts f on a random input.

Definition 1 (One-Way Trapdoor Permutation)**.** *Let $\Theta = (\mathcal{KG}, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^n$ and let k, ℓ be such that $k + \ell \le n$. The probability of an algorithm $\mathcal{I}$ in q-inverting Θ on bits $[k+1 \dots k+\ell]$ is*

$$\mathbf{Succ}_{\Theta}^{\textsf{s-pd-OW}}(k, \ell, q, \mathcal{I}) \stackrel{\text{def}}{=} \Pr\left[\mathsf{OW}_k^\ell : [s]_k^\ell \in S \wedge |S| \le q\right]$$

where OW_k^ℓ is the experiment:

$$(pk, sk) \leftarrow \mathcal{KG};\ s \xleftarrow{\$} \{0,1\}^n;\ S \leftarrow \mathcal{I}(f_{pk}(s), pk)$$

This definition generalizes the set partial-domain one-wayness problem of Fujisaki et al. [21], which corresponds to the case $k = 0$. We define $\mathbf{Succ}_{\Theta}^{\textsf{s-pd-OW}}(k, \ell, q, t)$ 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 $\mathbf{Succ}_{\Theta}^{\mathsf{OW}}(t)$.

2.2 Security of Padding-Based Encryption Schemes

We consider two security notions for public-key encryption: chosen-plaintext security, or $\mathsf{IND\text{-}CPA}$, and adaptive chosen-ciphertext security, or $\mathsf{IND\text{-}CCA}$. Both can be described succinctly using the following experiment, or game, in which an adversary $\mathcal{A}$ represented as a pair of procedures $(\mathcal{A}_1, \mathcal{A}_2)$ interacts with a challenger:

$$\begin{array}{l} (pk, sk) \leftarrow \mathcal{KG}; \\ (m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}}(pk); \\ b \xleftarrow{\$} \{0,1\}; \\ c^\star \leftarrow \mathcal{E}_{pk}(m_b); \\ \bar{b} \leftarrow \mathcal{A}_2^{\vec{H}}(c^\star, \sigma) \end{array}$$

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 $c^\star$ (the challenge) to the adversary. Finally, the adversary outputs a guess $\bar{b}$ for b. Note that $\mathcal{A}_1$ and $\mathcal{A}_2$ can communicate with each other through σ 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 $\mathcal{A}$ is given access to a decryption oracle $\mathcal{D}_{sk}(\cdot)$, with the proviso that $\mathcal{A}_2$ cannot ask for the decryption of the challenge ciphertext $c^\star$. In the remainder, we note $\boldsymbol{L}_H$ the list of queries that the adversary makes to a hash oracle H in either experiment, and $\boldsymbol{L}_{\mathcal{D}}$ the list of decryption queries that $\mathcal{A}_2$ 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 (Adversary Advantage). *The advantage of $\mathcal{A}$ against the* CPA *and* CCA *security of an encryption scheme $\Pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is defined respectively as:*

$$\mathbf{Adv}_{\Pi}^{\mathsf{CPA}}(\mathcal{A}) \stackrel{\text{def}}{=} 2 \Pr\left[\mathsf{CPA} : \bar{b} = b\right] - 1$$
$$\mathbf{Adv}_{\Pi}^{\mathsf{CCA}}(\mathcal{A}) \stackrel{\text{def}}{=} 2 \Pr\left[\mathsf{CCA} : \bar{b} = b\right] - 1$$

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

To define asymptotic security, we consider instead of a single scheme, a family of schemes Π_η indexed by a security parameter $\eta \in \mathbb{N}$. The above probability quantities thus become functions of η. We say that a scheme is secure if the advantage of any adversary executing in time polynomial in η is a negligible function ν of η, 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 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 [12] as a running example. RSA-OAEP, which instantiates OAEP with RSA as trapdoor permutation is recommended by several international standards.

Definition 3 (OAEP). *Let $(\mathcal{KG}_f, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^n$, $\rho \in \mathbb{N}$, and let*

$$G : \{0,1\}^k \to \{0,1\}^\ell \qquad H : \{0,1\}^\ell \to \{0,1\}^k$$

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

$$\begin{array}{lcl}
\mathcal{KG} & \stackrel{\text{def}}{=} & (pk, sk) \leftarrow \mathcal{KG}_f;\ \mathsf{return}\ (pk, sk) \\
\mathcal{E}_{pk}(m) & \stackrel{\text{def}}{=} & r \xleftarrow{\$} \{0,1\}^k;\ s \leftarrow G(r) \oplus (m \,\|\, 0^\rho);\ t \leftarrow H(s) \oplus r;\ \mathsf{return}\ f_{pk}(s \,\|\, t) \\
\mathcal{D}_{sk}(c) & \stackrel{\text{def}}{=} & s \,\|\, t \leftarrow f_{sk}^{-1}(c);\ r \leftarrow t \oplus H(s);\ m \leftarrow s \oplus G(r);\ \mathsf{if}\ [m]_{\ell-\rho}^{\rho} = 0^\rho\ \mathsf{then\ return}\ [m]_0^{\ell-\rho}\ \mathsf{else\ return}\ \bot
\end{array}$$

Additionaly, we use PSS-E, a variant of OAEP introduced by Coron et al. [17], as a running example throughout the paper.

Definition 4 (PSS-E). *Let $(\mathcal{KG}_f, f, f^{-1})$ be a family of trapdoor permutations on $\{0,1\}^k$, $k_0 \in \mathbb{N}$, and let*

$$G : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1} \qquad H : \{0,1\}^{k-k_1} \to \{0,1\}^{k_1}$$

be two hash functions such that $k_1 < k$. PSS-E is composed of the following triple of algorithms:

$$\begin{array}{lcl}
\mathcal{KG} & \stackrel{\text{def}}{=} & (pk, sk) \leftarrow \mathcal{KG}_f;\ \mathsf{return}\ (pk, sk) \\
\mathcal{E}_{pk}(m) & \stackrel{\text{def}}{=} & r \xleftarrow{\$} \{0,1\}^{k_0};\ s \leftarrow H(m \,\|\, r);\ t \leftarrow G(s) \oplus (m \,\|\, r);\ \mathsf{return}\ f_{pk}(s \,\|\, t) \\
\mathcal{D}_{sk}(c) & \stackrel{\text{def}}{=} & s \,\|\, t \leftarrow f_{sk}^{-1}(c);\ x \leftarrow t \oplus G(s);\ \mathsf{if}\ s = H(x)\ \mathsf{then\ return}\ [x]_0^{k-k_1-k_0}\ \mathsf{else\ return}\ \bot
\end{array}$$

3 Symbolic Tools

We introduce algebraic expressions to model padding-based encryption schemes, and symbolic notions to model the knowledge that either an adversary or an honest user can derive from an expression.

3.1 Syntax and Semantics of Expressions

Expressions are built from a set $\mathcal{R}$ of random bitstrings and a set $\mathcal{X}$ of variables, using bitstring operations, hash functions drawn from a set $\mathcal{H}$, and trapdoor permutations drawn from a set $\mathcal{F}$. The grammar of expressions is

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

where $x \in \mathcal{X}$, $r \in \mathcal{R}$, $n, \ell \in \mathcal{S}$, $\mathsf{f} \in \mathcal{F}$, 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:

$$\mathsf{f}(G(r) \oplus (m \,\|\, 0) \,\|\, H(G(r) \oplus (m \,\|\, 0)) \oplus r).$$

Expressions are typed using a size-based type system. A type $\ell \in \mathcal{S}$ 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 $e_1 \,\|\, e_2$ is $|e_1| + |e_2|$ and $e_1 \oplus e_2$ is well-typed iff $|e_1| = |e_2|$. In an asymptotic setting, we interpret size variables as functions that grow polynomially with the security parameter.

We let $\mathcal{R}(e)$, $\mathcal{X}(e)$, $\mathcal{F}(e)$ and $\mathcal{H}(e)$ denote, respectively, the sets of random bitstrings, variables, trapdoor permutation, and hash function symbols occurring in an expression e. We note $e\,\{\vec{e}_1/\vec{e}_0\}$ the simultaneous substitution of $\vec{e}_0$ by $\vec{e}_1$ in e and $\mathcal{ST}_H(e)$ the set of sub-expressions of e that are of the form $H(e')$. Abusing notation, we write e.g. $\mathcal{R}(e_1, e_2)$ instead of $\mathcal{R}(e_1) \cup \mathcal{R}(e_2)$.

Given values for size variables in $|e|$ and trapdoor permutations with adequate domains, we interpret a well-typed expression e as a probabilistic algorithm $(\!|e|\!)$ that calls oracles in $\mathcal{H}(e)$ and takes as parameters values for variables in $\mathcal{X}(e)$. The algorithm is implicitly parametrized by public and secret keys (if required) of the trapdoor permutations that occur in e. See Appendix A.1 for a formal definition.

3.2 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 1. The relation $=_E$ is sound [10] for all trapdoor permutations and valid key pairs. This means that for all closed expressions s and t, $s =_E t$ implies that $(\!|s|\!)$ and $(\!|t|\!)$ return the same value when shared random bitstrings are jointly sampled; formally,

$$\Pr\left[y \leftarrow (\!|s \,\|\, t|\!) : [y]_0^{|s|} = [y]_{|s|}^{|t|}\right] = 1.$$

$$(e_1 \oplus e_2) \oplus e_3 = e_1 \oplus (e_2 \oplus e_3) \qquad e_1 \oplus e_2 = e_2 \oplus e_1 \qquad e \oplus 0^{|e|} = e \qquad e \oplus e = 0^{|e|}$$

$$[0^n]_\ell^k = 0^k \qquad (e_1 \,\|\, e_2) \,\|\, e_3 = e_1 \,\|\, (e_2 \,\|\, e_3) \qquad [e_1 \,\|\, e_2]_0^{|e_1|} = e_1 \qquad [e_1 \,\|\, e_2]_{|e_1|}^\ell = [e_2]_0^\ell$$

$$[e_1 \oplus e_2]_n^\ell = [e_1]_n^\ell \oplus [e_2]_n^\ell \qquad [e]_0^{|e|} = e \qquad [[e]_n^\ell]_{n'}^{\ell'} = [e]_{n+n'}^{\ell'} \qquad [e]_n^\ell \,\|\, [e]_{n+\ell}^{\ell'} = [e]_n^{\ell+\ell'}$$

$$\mathsf{f}(\mathsf{f}^{-1}(e)) = e \qquad \mathsf{f}^{-1}(\mathsf{f}(e)) = e$$

Figure 1: Equational theory for algebraic expressions

We use the standard notion of deducibility from symbolic cryptography to reason about adversarial knowledge. Informally, an expression e' is deducible from e if there exists an efficient algorithm that computes e' from e. We use contexts to describe such algorithms symbolically. A context is an expression with a distinguished variable $*$. The application $C[e]$ of a context C to an expression e is defined by substitution.

Definition 5 (Deducibility)**.** *An expression e' is deducible from another expression e, written $e \vdash e'$, if there is a context C such that*

1. $\mathcal{R}(C) = \emptyset$, $\mathcal{X}(C) \subseteq \{*\}$;
2. f^{-1} *does not occur in* C, *and*
3. $C[e] =_E e'$.

We define $\vdash^\star$ analogously, but dropping restriction 2.

In Appendix A.1 we prove that $\vdash$ is sound in the following sense: $e \vdash e'$ implies that there is an efficient algorithm that computes e' from e and the public keys of the trapdoor permutations in e and e'. For $\vdash^\star$ we obtain an identical result, except that the corresponding algorithm can also use the secret keys of the trapdoor permutations. Note that we do not require $\vdash$-completeness, also known as $\nvdash$-soundness [10], since we never use the deducibility relation to conclude that an expression is not computable from another.

3.3 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 $C[\cdot] =_E C'[\cdot]$ succeed for e_0 if and only if they succeed for e_1. Since the completeness of $=_E$ with respect to the computational interpretation of expressions is an open question, 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 $\sharp$ in place of $\neq_E$. The soundness of this relation implies that if $s \sharp t$ holds, then the probability that the results of $(\!|s|\!)$ and $(\!|t|\!)$ coincide when shared random bitstrings are jointly sampled is negligible. For a given set of axioms Γ, the relation $\sharp$ is inductively defined by the rules given in Figure 2.

We use axioms of the form

$$\Gamma \subseteq \{s \sharp t \mid s, t \in \{0\} \cup \mathcal{X}\}$$

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

$$\frac{s \sharp t \in \Gamma}{s \sharp t}[\mathsf{Ax}] \qquad \frac{C[s] \sharp C[t]}{s \sharp t}[\mathsf{Ctx}] \qquad \frac{r \notin \mathcal{R}(s)}{r \sharp s}[\mathsf{Rnd}] \qquad \frac{s \sharp u \quad t =_E u}{s \sharp t}[\mathsf{EqE}]$$

$$\frac{e \sharp e_1 \quad \cdots \quad e \sharp e_k \quad H \notin \mathcal{H}(s)}{H(e) \sharp (\oplus_{i=1}^{k} H(e_i)) \oplus s}[\mathsf{Hash}]$$

Figure 2: Proof rules for apartness $\sharp$

Definition 6 (Static Distinguishability)**.** *The relation $\not\approx$ is the smallest symmetric relation on well-typed expressions such that $e_0 \not\approx e_1$ iff there are contexts C and C' such that*

1. $\mathcal{R}(C, C') = \emptyset$, $\mathcal{X}(C) \subseteq \{*\}$;
2. f^{-1} *does not occur in* C, C', *and*
3. $C[e_0] =_E C'[e_0]$ *and* $C[e_1] \sharp C'[e_1]$.

We use static distinguishability to find attacks on encryption schemes. For the attacks to be meaningful, we rely on the soundness of $\not\approx$, which follows from the soundness of $=_E$ and $\sharp$. Informally, if $e_0 \not\approx e_1$ holds, then there is an efficient algorithm distinguish that when given as input the public keys of trapdoor permutations and a value computed using $(\!|e_0|\!)$ returns 0, but when given instead a value computed using $(\!|e_1|\!)$ returns 1, except with negligible probability.

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

$$C = [*]_0^{|\mathsf{f}(r)|} \text{ and } C' = \mathsf{f}\left([*]_{|\mathsf{f}(r)|}^{|r|}\right)$$

it holds that $C[e_0] =_E C'[e_0]$ and $C[e_1] \sharp C'[e_1]$. 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.

4 Proof Systems

This section introduces logics for proving chosen-plaintext and chosen-ciphertext 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 (§ 4.1 and § 4.2) from the computation of concrete security bounds from valid derivations (§ 4.3). The soundness of the logics is stated in § 4.4 and proven in the Appendix.

4.1 Chosen-Plaintext Security

Judgments predicate over the probability of an event ϕ in the CPA experiment where the challenge ciphertext is computed using $(\!|c^\star|\!)$, which we denote $\mathsf{CPA}(c^\star)$. Events are drawn from the grammar

$$\phi ::= \mathsf{Guess} \mid \mathsf{Ask}(H, e) \mid \phi \wedge \phi$$

where Guess corresponds to the adversary correctly guessing the hidden bit b, and $\mathsf{Ask}(H, e)$ corresponds to the adversary asking the query $H(e)$.

$$\frac{\vDash_{\hat{p}} c^\star : \phi \qquad r \notin \mathcal{R}(e)}{\vDash_{\hat{p}} c^\star \{e \oplus r/r\} : \phi\{e \oplus r/r\}}[\mathsf{Opt}] \qquad \frac{\vDash_{\hat{p}} c^\star : \phi}{\vDash_{\hat{p}} c^\star \{\mathsf{f}(r)/r\} : \phi\{\mathsf{f}(r)/r\}}[\mathsf{Perm}] \qquad \frac{\vDash_{\hat{p}} c_2^\star : \phi_2 \quad c_1^\star =_E c_2^\star \quad \phi_1 \Longrightarrow_E \phi_2}{\vDash_{\hat{p}} c_1^\star : \phi_1}[\mathsf{Sub}]$$

$$\frac{\vDash_{\hat{p}} c^\star \{r_1 \,\|\, r_2/r\} : \phi\{r_1 \,\|\, r_2/r\} \quad r_1, r_2 \notin \mathcal{R}(c^\star, \phi) \quad r_1 \neq r_2}{\vDash_{\hat{p}} c^\star : \phi}[\mathsf{Split}] \qquad \frac{\vDash_{\hat{p}} c^\star : \phi \quad r_1, r_2 \notin \mathcal{R}(c^\star, \phi) \quad r_1 \neq r_2}{\vDash_{\hat{p}} c^\star \{r_1 \,\|\, r_2/r\} : \phi\{r_1 \,\|\, r_2/r\}}[\mathsf{Merge}]$$

$$\frac{\vDash_{\hat{p}} c^\star : \phi \quad \vDash_0 c^\star : \mathsf{Ask}(H, e) \quad r \notin \mathcal{R}(e) \quad H \notin \mathcal{H}(c^\star, \phi, e)}{\vDash_{\hat{p}} c^\star \{H(e)/r\} : \phi\{H(e)/r\}}[\mathsf{Fail}_1]$$

$$\frac{\vDash_{\hat{p}} c^\star : \phi \quad \vDash_0 c^\star \{H(e)/r\} : \phi\{H(e)/r\} \wedge \mathsf{Ask}(H, e) \quad r \notin \mathcal{R}(e) \quad H \notin \mathcal{H}(c^\star, \phi, e)}{\vDash_{\hat{p}} c^\star \{H(e)/r\} : \phi\{H(e)/r\}}[\mathsf{Fail}_2]$$

$$\frac{\mathcal{X}(c^\star) = \emptyset}{\vDash_{1/2} c^\star : \mathsf{Guess}}[\mathsf{Ind}] \qquad \frac{\vec{e} \,\|\, \mathcal{R}(c^\star) \,\|\, m \vdash^\star r \qquad r \notin \mathcal{R}(c^\star)}{\vDash_0 c^\star : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)}[\mathsf{Rnd}(|r|, \vec{H})]$$

$$\frac{\vec{e} \,\|\, r_2 \,\|\, m \vdash [r_1]_k^\ell \quad \mathsf{f}(r_1) \,\|\, r_2 \,\|\, m \vdash c^\star \quad r_1 \neq r_2 \quad (\mathsf{f}, k, \ell) \in \Gamma}{\vDash_0 c^\star : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)}[\mathsf{OW}_k^\ell(\mathsf{f}, \vec{H})]$$

Figure 3: Proof rules of the CPA logic ($\Longrightarrow_E$ denotes implication in first-order logic with respect to $=_E$)

Judgments are of the form $\vDash_{\hat{p}} c^\star : \phi$, where $c^\star$ is a well-typed expression with $\mathcal{X}(c^\star) \subseteq \{m\}$, ϕ is an event, and $\hat{p} \in \{0, 1/2\}$ is a probability tag[1]. The CPA logic is designed to ensure that if $\vDash_{\hat{p}} c^\star : \phi$ is derivable, then the probability of ϕ in the experiment $\mathsf{CPA}(c^\star)$ is upper bounded by $\hat{p}$ plus a negligible term. In particular, if $\vDash_{1/2} c^\star : \mathsf{Guess}$ is derivable, then any scheme Π with encryption algorithm $(\!|c^\star|\!)$ is asymptotically secure against chosen-plaintext attacks.

Derivability in the logic is parametrized by a set Γ of computational assumptions of the form (f, k, ℓ), where $\mathsf{f} \in \mathcal{F}$ and $k, \ell \in \mathcal{S}$. Such assumptions state that it is hard to compute $[r]_k^\ell$ from $\mathsf{f}(r)$. There is no restriction on Γ: it may contain assumptions for several permutations, and multiple assumptions for any given permutation.

Figure 3 presents the rules of the proof system. Note that the terminal rules [Rnd] and [OW] are decorated with information that is needed to compute concrete bounds (§ 4.3). We only allow to apply a rule if all instances of expressions are well-typed; this implies that for any valid derivation ∇ there exists a common typing environment for all rule applications.

The rules [Opt], [Perm], [Merge], and [Split] correspond to bridging steps in proofs and allow reformulating judgments into equivalent forms. The rule [Opt] corresponds to *optimistic sampling*, which allows substituting $e \oplus r$ by r provided that $r \notin \mathcal{R}(e)$. The rule [Perm] replaces all occurrences of $\mathsf{f}(r)$ by r. The rule [Merge] replaces the concatenation of two random bitstrings r_1 and r_2 with a fresh random bitstring r. The rule [Split] performs the opposite transformation; it replaces a random bitstring r with the concatenation of two fresh random bitstrings r_1 and r_2.

Rule [Sub] can be used to weaken the event and replace expressions in judgments with equivalent expressions. This rule is often needed to prepare the ground for the application of another rule: e.g. rule

[1]Probability tags increase the readability of rules and support a more uniform interpretation of judgments. However, they are superfluous and can be deduced from the shape of the event: for every valid judgment $\vDash_{\hat{p}} c^\star : \phi$, the probability tag $\hat{p}$ is $1/2$ iff $\phi = \mathsf{Guess}$.

[Opt] can be applied to a judgment that contains $H(r)$ to obtain $H(e \oplus r)$ by first using [Sub] to replace $H(r)$ by $H(e \oplus (e \oplus r))$.

The rules [Fail_1] and [Fail_2], for *equivalence up to failure*, correspond to specialized forms of Shoup's Fundamental Lemma [31] and create two branches in a derivation. These rules can be used to substitute an expression $H(e)$ by a random bitstring r, incurring a probability loss in the reduction corresponding to the probability of the CPA adversary asking the query $H(e)$. One can choose to analyze this probability either after (as in rule [Fail_1]) or before applying the substitution (as in rule [Fail_2]).

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 a plaintext variable; 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 it is exactly $1/2$. The rule [Rnd] closes a branch in the case when the adversary must make a number of oracle queries that would 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: this is the case when the image of a random element (the challenge of an inverter) can be embedded in the challenge ciphertext of the CPA experiment in such a way that its (partial) pre-image can be computed from the oracle queries made by the CPA adversary. To apply this rule, r_1 is usually obtained by searching for applications of f in $c^\star$ and r_2 is set to the concatenation of all random bitstrings in $\mathcal{R}(c^\star) \setminus \{r_1\}$. Additionally, one must check that the assumption (f, k, ℓ) is an assumption in Γ.

4.2 Chosen-Ciphertext Security

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

$$\begin{array}{lcl} F & ::= & \mathsf{find}\ x\ \mathsf{in}\ \boldsymbol{L}_H, F \ \mid\ \diamond \\ T & ::= & e = e' \ \mid\ \mathsf{Ask}(H, e) \ \mid\ T \wedge T \\ D & ::= & F : T \rhd e \end{array}$$

We assume that D does not contain random bitstrings and that all variables in expressions in D are bound by find, except for a distinguished parameter c denoting the ciphertext queried to the oracle. The above grammar is sufficiently expressive to encode decryption algorithms of padding-based schemes, as well as plaintext-simulators used in reductions.[2]

Informally, given as input a ciphertext c, an algorithm $\mathsf{find}\ \vec{x}\ \mathsf{in}\ \boldsymbol{L}_{\vec{H}} : T \rhd e$ searches among the queries made by an adversary to oracles $\vec{H}$ for values $\vec{v}$ satisfying T. If such values are found, it returns the value of $e\,\{\vec{v}/\vec{x}\}$; otherwise, it returns $\perp$. Conditions in T are interpreted as equality checks on bitstrings and membership tests in the lists of oracle queries made by the adversary. We write $T \rhd e$ instead of $\diamond : T \rhd e$ and note $(\!|D|\!)$ the interpretation of D as an algorithm.

Events of the logic are as in the CPA logic or of the form $\exists x \in \boldsymbol{L}_{\mathcal{D}}.\ T$. Events Guess and $\mathsf{Ask}(H, e)$ are defined as in the CPA logic but for the CCA experiment. For $\exists x \in \boldsymbol{L}_{\mathcal{D}}.\ T$, the quantified variable ranges

[2] 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.

$$\dfrac{\models_{\hat{p}} c^\star : \phi \quad \mathcal{ST}_{\mathsf{f}^{-1}}(T,t) = \emptyset}{\models_{\hat{p}} (c^\star, F : T \rhd t) : \phi}[\mathsf{Pub}(F,T,t)] \qquad \dfrac{\models_{\hat{p}} (c^\star, \mathsf{find}\ x\ \mathsf{in}\ \boldsymbol{L}_H, F : T \wedge x = e \rhd t) : \phi}{\models_{\hat{p}} (c^\star, F : T\,\{e/x\} \wedge \mathsf{Ask}(H,e) \rhd t\,\{e/x\}) : \phi}[\mathsf{Find}]$$

$$\dfrac{\models_{\hat{p}} (c_2^\star, F : T_2 \rhd t_2) : \phi_2 \quad c_1^\star =_E c_2^\star \quad \phi_1 \Longrightarrow_E \phi_2 \quad T_1 \Longleftrightarrow_E T_2 \quad t_1 =_E t_2}{\models_{\hat{p}} (c_1^\star, F : T_1 \rhd t_1) : \phi_1}[\mathsf{Conv}] \qquad \dfrac{}{\models_0 (c^\star, D) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ c = c^\star}[\mathsf{False}]$$

$$\dfrac{\begin{array}{c}\mathcal{ST}_H(c^\star) = \{H(e^\star)\} \quad \mathcal{ST}_H(T,t) = \{H(e)\} \\ \models_{\hat{p}} (c^\star, T \wedge \mathsf{Ask}(H,e) \rhd t) : \phi \quad \models_0 (c^\star, T \wedge \mathsf{Ask}(H,e) \rhd t) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ T \wedge e^\star = e \quad \models \mathsf{negl}_r(T\,\{r/H(e)\})\end{array}}{\models_{\hat{p}} (c^\star, T \rhd t) : \phi}[\mathsf{Bad}]$$

$$\dfrac{\models \mathsf{negl}_r(T_i)}{\models \mathsf{negl}_r(T_1 \wedge T_2)}[\mathsf{PAnd}_i] \qquad \dfrac{c \,\|\, e \vdash^\star [r]_k^\ell \qquad r \notin \mathcal{R}(e')}{\models \mathsf{negl}_r(e = e')}[\mathsf{PEqs}(\ell)] \qquad \dfrac{c \,\|\, e \vdash^\star [r]_k^\ell}{\models \mathsf{negl}_r(\mathsf{Ask}(H,e))}[\mathsf{PRnd}(\ell,H)]$$

$$\dfrac{r \notin \mathcal{R}(c^\star, e)}{\models_0 c^\star : e = r}[\mathsf{Eqs}(|r|)] \qquad \dfrac{\vec{e} \,\|\, \mathcal{R}(c^\star) \,\|\, m \vdash^\star r \qquad r \notin \mathcal{R}(c^\star)}{\models_0 c^\star : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{Ask}(H_1,e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n,e_n)}[\mathsf{Rnd}(|r|,\vec{H})]$$

$$\dfrac{\vec{e} \,\|\, r_2 \,\|\, m \vdash [r_1]_k^\ell \quad \mathsf{f}(r_1) \,\|\, r_2 \,\|\, m \vdash c^\star \quad r_1 \neq r_2 \quad (\mathsf{f},k,\ell) \in \Gamma}{\models_0 c^\star : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{Ask}(H_1,e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n,e_n)}[\mathsf{OW}_k^\ell(\mathsf{f},\vec{H})]$$

Figure 4: Proof rules for CCA judgments, proof rules for tests, and extended rules for the CPA logic

over the ciphertexts queried to the decryption oracle during the second phase of the CCA experiment; tests are interpreted as described above.

A judgment $\models_{\hat{p}} (c^\star, D) : \phi$ predicates over the probability of ϕ in the CCA experiment where the challenge ciphertext is computed using $(\!|c^\star|\!)$, and decryption queries are answered by $(\!|D|\!)$; we note this experiment $\mathsf{CCA}(c^\star, D)$. The judgment states that the probability of ϕ in this experiment is upper bounded by $\hat{p}$ plus a negligible term.

Figure 4 presents the rules of the logic. The rule [Bad] allows to transform the decryption oracle so that it rejects ciphertexts whose decryption requires to make oracle 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 4.

The rule [False] captures the fact that the adversary cannot ask for the decryption of $c^\star$ 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.

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.

$$\mathcal{B}_{(t_{\mathcal{A}},\vec{q})}(\nabla) = \begin{cases} \mathcal{B}(\nabla_1)+\mathcal{B}(\nabla_2) & \text{if } L_\nabla = [\mathsf{Fail}_{1,2}] \\ \mathcal{B}(\nabla_1)+\mathcal{B}(\nabla_2)+q_{\mathcal{D}}\,\mathcal{B}(\nabla_3) & \text{if } L_\nabla = [\mathsf{Bad}] \\ \mathcal{B}_{(t_{\mathcal{S}},\vec{q}^{\mathcal{S}})}(\nabla_1) & \text{if } L_\nabla = [\mathsf{Pub}(F,T,u)] \\ 1/2 & \text{if } L_\nabla = [\mathsf{Ind}] \\ 0 & \text{if } L_\nabla = [\mathsf{False}] \\ q_{\vec{H}} \times 2^{-\ell} & \text{if } L_\nabla = [\mathsf{Rnd}(\ell,\vec{H})] \\ q_H \times 2^{-\ell} & \text{if } L_\nabla = [\mathsf{PRnd}(\ell,H)] \\ 2^{-\ell} & \text{if } L_\nabla = [\mathsf{Eqs}(\ell)] \\ 2^{-\ell} & \text{if } L_\nabla = [\mathsf{PEqs}(\ell)] \\ \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta_{\mathsf{f}}}(k,\ell,q_{\vec{H}},t_{\mathcal{I}}) & \text{if } L_\nabla = [\mathsf{OW}^\ell_k(\mathsf{f},\vec{H})] \\ \mathcal{B}(\nabla_1) & \text{otherwise} \end{cases}$$

where $t_{\mathcal{I}} = t_{\mathcal{A}} + T(C_2) + q_{\vec{H}} \times T(C_1)$, where C_1, C_2 are the contexts witnessing the first and second deducibility conditions in the premise of $[\mathsf{OW}^\ell_k(\mathsf{f},\vec{H})]$, $q_{\vec{H}} = \prod_{H_i \in \vec{H}} q_{H_i}$, and

$$t_{\mathcal{S}} = t_{\mathcal{A}} + q_{\mathcal{D}} \times (\mathsf{T}(u) + \mathsf{T}(T) \times \prod_{H_j \text{ in } F} q_{H_j}) \qquad q^{\mathcal{S}}_{H_i} = q_{H_i} + q_{\mathcal{D}} \times (\mathsf{Q}_i(u) + \mathsf{Q}_i(T) \times \prod_{H_j \text{ in } F} q_{H_j})$$

Figure 5: Computation of concrete security bounds from logical derivations

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 $\vDash_{\hat{p}} c^\star : \phi$, where ϕ 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 4.

4.3 Concrete Bounds

In this section we show how to extract concrete security bounds from derivations in our logics. Security bounds p are drawn from the grammar

$$p ::= \epsilon \mid 1/2 + \epsilon \quad q ::= 1 \mid q \times q \mid q_H \mid q_{\mathcal{D}} \quad t ::= t_{\mathcal{A}} \mid q \mid q \times t_{\mathsf{f}} \mid t + t$$
$$\epsilon ::= 0 \mid 2^{-k} \mid \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta_{\mathsf{f}}}(k,\ell,q,t) \mid q \times \epsilon \mid \epsilon + \epsilon$$

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

Security bounds are computed from a derivation ∇ in the CPA or CCA logic using a function $\mathcal{B}$ parametrized by the computational resources of an adversary, namely, its execution time $t_{\mathcal{A}}$ and the number of oracle queries it can make, $\vec{q}$. The definition of function $\mathcal{B}$ is given in Figure 5. In this figure, we omit resource parameters when they remain unchanged. Moreover, for a derivation ∇ with rule R at its root, we use L_∇ to refer to R's label and ∇_i to refer to the derivation of the i-th premise of R (premises are enumerated from left to right).

For all bridging rules, the bound is inherited from their single premise. For rules [Fail_1] and [Fail_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. Rule [Bad] is similar except that bounds come from the probability of two different failure events, and one can be triggered in any decryption query.

The case of rule [Pub] represents a reduction from CCA to CPA, where a simulator $\mathcal{S}$ uses a CCA adversary $\mathcal{A}$ to win in the CPA experiment. We therefore have to compute bounds for the computational resources $(t_{\mathcal{S}}, \vec{q}^{\mathcal{S}})$ used by $\mathcal{S}$ in terms of the resources of $\mathcal{A}$ 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, $t_{\mathcal{S}}$ can be defined as $t_{\mathcal{A}} + q_{\mathcal{D}} \times t_{\mathcal{D}}$ where $t_{\mathcal{D}}$ 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 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 $q^{\mathcal{S}}_{H_i}$.

The rules [Ind] and [False], yield exact bounds that can be readily used. The cases of rules [Rnd], and [PRnd], [Eqs], [PEqs], correspond to the probability of guessing a random bitstring of length ℓ in $q_{\vec{H}}$, q_H, or just 1 tries. In the case of rule [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 $\ell = |\mathsf{f}|$, we can alternatively use the standard reduction from set one-wayness to one-wayness to obtain the bound $\mathbf{Succ}^{\mathsf{OW}}_{\Theta_{\mathsf{f}}}(t_{\mathcal{A}} + q_{\vec{H}} + q_{\vec{H}} \times t_{\mathsf{f}})$. Here, the adjusted time bound accounts for the fact that the inverter inspects every combination of queries made to $\vec{H}$, computes a value, applies f to that value and compares the result to its challenge to find the right pre-image.

4.4 Soundness

Let Π be an encryption scheme with encryption algorithm $(\!|c^\star|\!)$ and decryption algorithm $(\!|D|\!)$. Assume that the interpretations of trapdoor permutations satisfy all the assumptions in Γ.

Theorem 7. *If ∇ is a derivation of $\vDash_{1/2} c^\star : \mathsf{Guess}$ under Γ, then*

$$\mathbf{Adv}^{\mathsf{CPA}}_{\Pi}(t_{\mathcal{A}}, \vec{q}) \leq 2\,\mathcal{B}_{(t_A,\vec{q})}(\nabla) - 1$$

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

Theorem 8. *If ∇ is a derivation of $\vDash_{1/2} (c^\star, D) : \mathsf{Guess}$ under Γ, then*

$$\mathbf{Adv}^{\mathsf{CCA}}_{\Pi}(t_{\mathcal{A}}, \vec{q}) \leq 2\ \mathcal{B}_{(t_A,\vec{q})}(\nabla) - 1$$

Moreover, this bound is negligible if $t_{\mathcal{A}}$, all q_H in $\vec{q}$, and $q_{\mathcal{D}}$ are polynomial in the security parameter, and all assumptions in Γ hold.

The proofs of these theorems rest on showing the soundness of each individual rule using game-based techniques. The proofs appear in the Appendix.

4.5 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.

$\underline{\nabla_{\mathsf{OW}}:}$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vDash_0 \mathsf{f}(r' \| r) : \mathsf{Ask}(G, r'')}\,[\mathsf{Rnd}(k,G)]}{\vDash_0 \mathsf{f}(r' \| r) : \mathsf{Ask}(G, r'' \oplus r)}\,[\mathsf{Opt}] \qquad \dfrac{\dfrac{}{\vDash_0 \mathsf{f}(r'') : \mathsf{Ask}(H, [r'']_0^{\ell}) \wedge \mathsf{Ask}(G, H([r'']_0^{\ell}) \oplus [r'']_{\ell}^{k})}\,[\mathsf{OW}_0^n(\mathsf{f}, (H,G))]}{\vDash_0 \mathsf{f}(r' \| r) : \mathsf{Ask}(H, r') \wedge \mathsf{Ask}(G, H(r') \oplus r)}\,[\mathsf{Merge}]}{\vDash_0 \mathsf{f}(r' \| r) : \mathsf{Ask}(G, H(r') \oplus r)}\,[\mathsf{Fail}_2]}{\vDash_0 \mathsf{f}(r' \| H(r') \oplus r) : \mathsf{Ask}(G, r)}\,[\mathsf{Opt}]$$

$\underline{\nabla_{\mathsf{PDOW}}:}$

$$\dfrac{\dfrac{\dfrac{}{\vDash_0 \mathsf{f}(r' \| r'') : \mathsf{Ask}(G, r)}\,[\mathsf{Rnd}(k,G)]}{\vDash_0 \mathsf{f}(r' \| r'' \oplus r) : \mathsf{Ask}(G, r)}\,[\mathsf{Opt}] \qquad \dfrac{\dfrac{\dfrac{}{\vDash_0 \mathsf{f}(r) : \mathsf{Ask}(H, [r]_0^{\ell})}\,[\mathsf{OW}_0^{\ell}(\mathsf{f}, H)]}{\vDash_0 \mathsf{f}(r' \| r) : \mathsf{Ask}(H, r')}\,[\mathsf{Merge}]}{\vDash_0 \mathsf{f}(r' \| r'' \oplus r) : \mathsf{Ask}(H, r')}\,[\mathsf{Opt}]}{\vDash_0 \mathsf{f}(r' \| H(r') \oplus r) : \mathsf{Ask}(G, r)}\,[\mathsf{Fail}_1]$$

$\underline{\nabla_{\mathsf{OAEP}}:}$

$$\dfrac{\dfrac{\dfrac{}{\vDash_{1/2} \mathsf{f}(r' \| H(r') \oplus r) : \mathsf{Guess}}\,[\mathsf{Ind}]}{\vDash_{1/2} \mathsf{f}(r' \oplus (m \| 0) \| H(r' \oplus (m \| 0)) \oplus r) : \mathsf{Guess}}\,[\mathsf{Opt}] \qquad \dfrac{\dfrac{\nabla}{\vDash_0 \mathsf{f}(r' \| H(r') \oplus r) : \mathsf{Ask}(G, r)}}{\vDash_0 \mathsf{f}(r' \oplus (m \| 0) \| H(r' \oplus (m \| 0)) \oplus r) : \mathsf{Ask}(G, r)}\,[\mathsf{Opt}]}{\vDash_{1/2} \mathsf{f}((G(r) \oplus (m \| 0)) \| H(G(r) \oplus (m \| 0)) \oplus r) : \mathsf{Guess}}\,[\mathsf{Fail}_1]$$

Figure 6: Derivations for CPA security of OAEP under one-wayness and partial-domain one-wayness

4.5.1 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 6. This first part proceeds by applying [Fail_1] to replace the expression $G(r)$ with a fresh random bitstring r'. To close the branch corresponding to the Guess event, we apply optimistic sampling, replacing $r' \oplus (m \| 0)$ by r'. In the judgment thus obtained, the challenge ciphertext does not depend on the plaintext, so we conclude by applying [Ind]. We continue with the branch corresponding to the failure event $\mathsf{Ask}(G, r)$ applying optimistic sampling. 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 6. The proof proceeds by applying rule [Fail_1] to replace $H(r')$ by a fresh random bitstring r''. The rule [Opt] is then applied in both premises to replace $r'' \oplus r$ by just r. The branch corresponding to the event $\mathsf{Ask}(G, r)$ can be closed by applying rule [Rnd] because the challenge ciphertext does not depend on r anymore. The branch corresponding to the event $\mathsf{Ask}(H, r')$ is closed by merging the random bitstrings r' and r as r and then reformulating the event as $\mathsf{Ask}(H, [r]_0^{\ell})$. At this point, it is clear that querying H on the ℓ most significant bits of r amounts to inverting f in its ℓ most significant bits, so we conclude applying [$\mathsf{OW}_0^{\ell}(\mathsf{f}, H)$].

The derivation for OW is depicted at the top of Figure 6. It proceeds by applying rule [Opt] to move the expression $H(r')$ from the challenge expression to the event, that becomes $\mathsf{Ask}(G, H(r') \oplus r)$. We then apply [$\mathsf{Fail}_2$] to replace $H(r')$ by a fresh random bitstring r''. Note that this rule performs a substitution only on one of the branches. The branch corresponding to event $\mathsf{Ask}(G, r'' \oplus r)$ is closed using rule [Opt] to replace $r'' \oplus r$ by r'', so that the event no longer depends on the challenge ciphertext and we can conclude using rule [Rnd]. Finally, the last branch is closed by merging $r' \| r$ into a fresh random bitstring r''. We

∇_{Ask} :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vDash_0 \mathsf{f}(r) : \mathsf{Ask}(G, [r]_0^{k_1})}\,[\mathsf{OW}_0^{k_1}(\mathsf{f}, G)]}{\vDash_0 \mathsf{f}(r'' \| r') : \mathsf{Ask}(G, r'')}\,[\mathsf{Merge}] \qquad \dfrac{}{\vDash_0 \mathsf{f}(r'' \| r') : \mathsf{Ask}(H, m \| r)}\,[\mathsf{Rnd}(k_0, H)]}{\vDash_0 \mathsf{f}(H(m \| r) \| r') : \mathsf{Ask}(G, H(m \| r))}\,[\mathsf{Fail}_1]}{\vDash_0 \mathsf{f}(H(m \| r) \| r' \oplus (m \| r)) : \mathsf{Ask}(G, H(m \| r))}\,[\mathsf{Opt}]$$

$\nabla_{\mathsf{PSS\text{-}E}}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vDash_{1/2} \mathsf{f}(r'' \| r') : \mathsf{Guess}}\,[\mathsf{Ind}] \qquad \dfrac{}{\vDash_0 \mathsf{f}(r'' \| r') : \mathsf{Ask}(H, m \| r)}\,[\mathsf{Rnd}(k_0, H)]}{\vDash_{1/2} \mathsf{f}(H(m \| r) \| r') : \mathsf{Guess}}\,[\mathsf{Fail}_1]}{\vDash_{1/2} \mathsf{f}(H(m \| r) \| r' \oplus (m \| r)) : \mathsf{Guess}}\,[\mathsf{Opt}] \qquad \dfrac{\nabla_{\mathsf{Ask}}}{\vDash_0 \mathsf{f}(H(m \| r) \| r' \oplus (m \| r)) : \mathsf{Ask}(G, H(m \| r))}\,[\mathsf{Opt}]}{\vDash_{1/2} \mathsf{f}(H(m \| r) \| G(H(m \| r)) \oplus (m \| r)) : \mathsf{Guess}}\,[\mathsf{Fail}_1]$$

Figure 7: A derivation for CPA security of PSS-E

reformulate the event accordingly and obtain

$$\mathsf{Ask}(H, [r'']_0^{\ell}) \wedge \mathsf{Ask}(G, H([r'']_0^{\ell}) \oplus [r'']_{\ell}^{k}).$$

If the adversary manages to query $[r'']_0^{\ell}$ to H and $H([r'']_0^{\ell}) \oplus [r'']_{\ell}^{k}$ to G, then the pre-image r'' of the challenge ciphertext can be computed from $\boldsymbol{L}_G$ and $\boldsymbol{L}_H$. Indeed, the ℓ most significant bits of r'' are a query in $\boldsymbol{L}_H$, while the remaining k bits are the exclusive-or of the answer to this query and some query in $\boldsymbol{L}_G$. Hence, a CPA adversary that triggers this event can be used to fully invert function f, and we can conclude by applying rule $[\mathsf{OW}_n^0(\mathsf{f}, (H, G))]$.

Concrete security bound. Applying the bound function $\mathcal{B}_{(t_{\mathcal{A}},(q_G,q_H))}$ to the derivation $\nabla_{\mathsf{OAEP}}, \nabla_{\mathsf{PDOW}}$ gives

$$1/2 + q_G \times 2^{-k} + \mathbf{Succ}_{\Theta}^{\mathsf{s\text{-}pd\text{-}OW}}(\ell, q_H, t_{\mathcal{A}} + q_H).$$

Applying the bound function $\mathcal{B}_{(t_{\mathcal{A}},(q_G,q_H))}$ to the derivation $\nabla_{\mathsf{OAEP}}, \nabla_{\mathsf{OW}}$ gives (after applying the standard reduction from set one-wayness to one-wayness)

$$1/2 + q_G \times 2^{-k} + \mathbf{Succ}_{\Theta}^{\mathsf{OW}}(t_{\mathcal{A}} + q_G\, q_H \times t_{\mathsf{f}}).$$

4.5.2 PSS-E

We prove the chosen-plaintext security and chosen-ciphertext security of PSS-E by exhibiting derivations in the CPA and CCA logics. Moreover, we illustrate how to derive concrete security bounds from these derivations. The encryption algorithm of PSS-E is given by the following expression

$$\mathsf{f}(H(m \| r) \| G(H(m \| r)) \oplus (m \| r))$$

Chosen-plaintext security. The derivation for chosen-plaintext security output by our tool is depicted in Figure 7. The proof starts by applying the rule $[\mathsf{Fail}_1]$, replacing the hash $G(H(m \| r))$ by a fresh random bitstring r'.

In the branch corresponding to the Guess event, we apply optimistic sampling, replacing $r' \oplus (m \,\|\, r)$ by r', and then [Fail_1] again, obtaining two premises. The premise corresponding to the original Guess event is proved using rule [Ind] since the challenge ciphertext no longer depends on m. The premise corresponding to the failure event can be discharged using [Rnd] because $m \,\|\, r \vdash^\star r$ and r does not appear in the challenge ciphertext, meaning that the adversary would have to guess r to trigger failure.

The derivation ∇ of the remaining premise of the first application of rule [Fail_1] is obtained using a similar strategy. We first apply [Opt] followed by [Fail_1], resulting in two premises:

- $\vDash_0 \mathsf{f}(r'' \,\|\, r') : \mathsf{Ask}(G, r'')$
- $\vDash_0 \mathsf{f}(r'' \,\|\, r') : \mathsf{Ask}(H, m \,\|\, r)$

To prove the first premise, observe that any adversary that asks r'' to G, can be used to invert f on its k_1 most significant bits. However, before applying rule OW, we must first apply [Merge] to replace $r'' \,\|\, r'$ by a fresh random variable r, so that the judgment has the appropriate form.

The second premise corresponds to the same situation encountered before, where the adversary must query $m \,\|\, r$ to H to trigger failure, but r does not occur in the challenge ciphertext. We conclude as before using Rnd.

Concrete security bound. Applying the bound function $\mathcal{B}_{(t_\mathcal{A},(q_G,q_H))}$ to the derivation $\nabla_{\mathsf{PSS\text{-}E}}$ yields

$$1/2 + 2q_H \times 2^{-k_0} + \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta}(k_1, q_G, t_\mathcal{A} + q_G).$$

Chosen-ciphertext security. The derivation for chosen-ciphertext security output by our tool is depicted in Figure 8. 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

$$\begin{array}{rcl} c^\star & = & \mathsf{f}(H(m \,\|\, r) \,\|\, G(H(m \,\|\, r)) \oplus (m \,\|\, r)) \\ D_1 & = & \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \rhd [x]_0^n \end{array}$$

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

$$D_2 \;=\; \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \wedge \mathsf{Ask}(H, x) \rhd [x]_0^n$$

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 $\vDash \mathsf{negl}_{r'}(\mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ s = r')$ and can be proven using [$\mathsf{PEqs}(k_1)$]. The second case corresponds to the premise

$$\vDash_0 (c^\star, D_2) : \exists c \in \boldsymbol{L}_\mathcal{D}.\, \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \wedge x = m \,\|\, r$$

and is proven using rules [False] and [Conv], because the event is equivalent to $\exists c \in \boldsymbol{L}_\mathcal{D}.\, c = c^\star$.

$\underline{\nabla_{\phi_2}:}$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vDash_0 \mathsf{f}(r'' \,\|\, r'_1 \,\|\, r'_2) : \phi_5}\,[\mathsf{Rnd}(k_0,H)]}{\vDash_0 \mathsf{f}(r'' \,\|\, r') : \phi_4}\,[\mathsf{Split}],[\mathsf{Opt}] \qquad \dfrac{}{\vDash_0 \mathsf{f}(r'' \,\|\, r') : \mathsf{Ask}(H, m \,\|\, r)}\,[\mathsf{Rnd}(k_0,H)]}{\dfrac{\vDash_0 \mathsf{f}(H(m \,\|\, r) \,\|\, r') : \phi_4}{\vDash_0 \mathsf{f}(H(m \,\|\, r) \,\|\, r' \oplus (m \,\|\, r)) : \phi_3}\,[\mathsf{Opt}],[\mathsf{Conv}]}\,[\mathsf{Fail}_1] \qquad \nabla_{\mathsf{Ask}}}{\vDash_0 \mathsf{f}(H(m \,\|\, r) \,\|\, G(H(m \,\|\, r)) \oplus (m \,\|\, r)) : \phi_2}\,[\mathsf{Fail}_1]$$

$\underline{\nabla_{\mathsf{Guess}}:}$

$$\dfrac{\dfrac{\dfrac{\nabla_{\mathsf{PSS\text{-}E}}}{\vDash_{1/2} (c^\star, D_4) : \mathsf{Guess}}\,[\mathsf{Pub}]}{\vDash_{1/2} (c^\star, D_3) : \mathsf{Guess}}\,[\mathsf{Find}] \qquad \dfrac{\dfrac{\nabla_{\phi_2}}{\vDash_0 (c^\star, D_4) : \phi_2}\,[\mathsf{Pub}]}{\vDash_0 (c^\star, D_3) : \phi_2}\,[\mathsf{Find}] \qquad \dfrac{}{\vDash \mathsf{negl}_{r'}(\mathsf{let}\ s \| t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{Ask}(H, r' \oplus t))}\,[\mathsf{PRnd}(k-k_1,H)]}{\vDash_{1/2} (c^\star, D_2) : \mathsf{Guess}}\,[\mathsf{Bad}],[\mathsf{PAnd}_2]$$

$\underline{\nabla^{\mathsf{CCA}}_{\mathsf{PSS\text{-}E}}:}$

$$\dfrac{\nabla_{\mathsf{Guess}} \qquad \dfrac{}{\vDash_0 (c^\star, D_2) : \phi_1}\,[\mathsf{Conv}],\,[\mathsf{False}] \qquad \dfrac{}{\vDash \mathsf{negl}_{r'}(\mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ s = r')}\,[\mathsf{PEqs}(k_1)]}{\vDash_{1/2} (c^\star, D_1) : \mathsf{Guess}}\,[\mathsf{Bad}]$$

$$\begin{aligned}
\phi_1 &\stackrel{\text{def}}{=} \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \wedge x = m \,\|\, r \\
\phi_2 &\stackrel{\text{def}}{=} \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \wedge \mathsf{Ask}(H,x) \wedge s = H(m \,\|\, r) \\
\phi_3 &\stackrel{\text{def}}{=} \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = r' \oplus t\ \mathsf{in}\ s = H(x) \wedge \mathsf{Ask}(H,x) \wedge s = H(m \,\|\, r) \\
\phi_4 &\stackrel{\text{def}}{=} \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{Ask}(H, r' \oplus (m \,\|\, r) \oplus t) \\
\phi_5 &\stackrel{\text{def}}{=} \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{Ask}(H, (r'_1 \oplus m \oplus [t]_0^n) \,\|\, r)
\end{aligned}$$

Figure 8: A derivation for CCA security of PSS-E

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

$$D_3 \quad = \quad \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \wedge \mathsf{Ask}(H,x) \wedge \mathsf{Ask}(G,s) \rhd [x]_0^n$$

The premise

$$\vDash \mathsf{negl}_{r'}(\mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = r' \oplus t\ \mathsf{in}\ s = H(x) \wedge \mathsf{Ask}(H,x))$$

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

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

$$D_4 \quad = \quad \mathsf{find}\ s, x\ \mathsf{in}\ \boldsymbol{L}_G, \boldsymbol{L}_H : c = \mathsf{f}(s \,\|\, G(s) \oplus x) \wedge s = H(x) \rhd [x]_0^n$$

Note that this decryption oracle is *public*, i.e. it does not use f^{-1} and can be efficiently simulated. Hence, we can apply rule [Pub] and proceed reasoning in the (extended) CPA logic. The branch corresponding to the original Guess event can be proven using the same derivation $\nabla_{\mathsf{PSS\text{-}E}}$ in Figure 7 that we used to prove CPA security.

We have only one outstanding goal,

$$\vDash_0 c^\star : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{let}\ x = G(s) \oplus t\ \mathsf{in}\ s = H(x) \wedge \mathsf{Ask}(H,x) \wedge s = H(m \,\|\, r)$$

To prove it, we first apply [Fail$_1$] to replace $G(H(m \,\|\, r))$ with a fresh random bitstring r'. The derivation ∇_{Ask} that bounds the probability of failure is as in the proof of CPA security in Figure 7. We apply rules

[Opt] and [Conv] to simplify the premise corresponding to the original event, obtaining

$$\vDash_0 \mathsf{f}(H(m \,\|\, r) \,\|\, r') : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\, \mathsf{let}\ s \,\|\, t = \mathsf{f}^{-1}(c)\ \mathsf{in}\ \mathsf{Ask}(H, r' \oplus (m \,\|\, r) \oplus t)$$

We then apply [Fail_1] again, this time to replace $H(m \,\|\, r)$ with a fresh random bitstring r''. The premise corresponding to the failure event can be readily proved using [$\mathsf{Rnd}(k_0, H)$] as in the proof of CPA security. To prove the remaining premise, observe that to trigger the event $\mathsf{Ask}(H, r' \oplus (m \,\|\, r) \oplus t)$, the adversary should be able to compute $[r' \oplus t]_n^{k_0} \oplus r$, but r is now independent of the challenge ciphertext. We formalize this reasoning by first splitting r' and then applying rule [Opt] to replace $[r' \oplus t]_n^{k_0} \oplus r$ by simply r. We conclude applying rule [Exists] and [$\mathsf{Rnd}(k_0, H)$].

Concrete security bound. The concrete security bound $\mathcal{B}_{(t_{\mathcal{A}},(q_G,q_H,q_{\mathcal{D}}))}(\nabla^{\mathsf{CCA}}_{\mathsf{PSS\text{-}E}})$ obtained from this proof is

$$\mathcal{B}_{(t_{\mathcal{S}},(q_G,q_H,q_{\mathcal{D}}))}(\nabla_{\mathsf{PSS\text{-}E}}) + \mathcal{B}_{(t_{\mathcal{S}},(q_G,q_H,q_{\mathcal{D}}))}(\nabla_{\phi_2}) + q_{\mathcal{D}} \times (2^{-k_1} + q_H \times 2^{k_1 - k})$$

where $t_{\mathcal{S}} = t_{\mathcal{A}} + q_{\mathcal{D}}\, q_G\, q_H \times t_{\mathsf{f}}$, and

$$\begin{aligned}
\mathcal{B}_{(t_{\mathcal{S}},(q_G,q_H,q_{\mathcal{D}}))}(\nabla_{\mathsf{PSS\text{-}E}}) &\approx 1/2 + 2\, q_H \times 2^{-k_0} + \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta}(k_1, q_G, t_{\mathcal{A}} + q_{\mathcal{D}}\, q_G\, q_H \times t_{\mathsf{f}}) \\
\mathcal{B}_{(t_{\mathcal{S}},(q_G,q_H,q_{\mathcal{D}}))}(\nabla_{\phi_2}) &= 3\, q_H \times 2^{-k_0} + \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta}(k_1, q_G, t_{\mathcal{A}} + q_{\mathcal{D}}\, q_G\, q_H \times t_{\mathsf{f}})
\end{aligned}$$

Observe that $t_{\mathcal{S}}$ comes from the two applications of rule [Find] and can be improved to $t_{\mathcal{A}} + q_{\mathcal{D}}\, q_H \times t_{\mathsf{f}}$, because to answer a decryption query, the simulator D_4 just needs to traverse $\boldsymbol{L}_H$ to find a value for x, the value of s is determined by $H(x)$. The overall bound is, ignoring constant factors,

$$1/2 + q_H \times 2^{-k_0} + q_{\mathcal{D}} \times 2^{-k_1} + q_{\mathcal{D}} \times q_H \times 2^{k_1 - k} + \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta}(k_1, q_G, t_{\mathcal{A}} + q_{\mathcal{D}}\, q_H \times t_{\mathsf{f}})$$

In comparison, the bound given by Coron et al. [17] is:

$$1/2 + q_H \times 2^{-k_0} + q_{\mathcal{D}} \times 2^{-k_1} + \mathbf{Succ}^{\mathsf{s\text{-}pd\text{-}OW}}_{\Theta}(k_1, q_G + q_H, t_{\mathcal{A}} + q_{\mathcal{D}}\, q_H \times t_{\mathsf{f}})$$

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 [Bad] in ∇_{Guess} in our reduction, eliminating the terms coming from ∇_{ϕ_2} and $q_{\mathcal{D}} \times q_H \times 2^{k_1 - k}$. The resulting bound, once adjusting q_G for the additional queries issued by H, coincides with that of Coron et al.

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.

5.1 Universal Attacks

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 $\mathsf{distinguish}$ that returns 0 for input $(\!|e\{m_0/m\}|\!)$ and 1 for input $(\!|e\{m_1/m\}|\!)$ with overwhelming probability. To mount an attack against CPA security using this algorithm, an adversary chooses plaintexts $(\!|m_0|\!)$ and $(\!|m_1|\!)$, receives the challenge ciphertext $c^\star$, and returns $\mathsf{distinguish}(c^\star)$. An example of a scheme vulnerable to this attack is a scheme with encryption algorithm given by

$$e = \mathsf{f}(r) \,\|\, (G(r) \oplus m) \,\|\, H(m)$$

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

To refute CCA security, we search for malleability attacks using deducibility. Specifically, we search for closed expressions m_0 and $\Delta \neq 0^{|m|}$ such that $e\{m_0/m\} \vdash e\{m_0 \oplus \Delta/m\}$; let C be the context witnessing the deducibility. This would imply an effective attack against CCA security: choose plaintexts m_0 and $m_1 \neq m_0$ and obtain the challenge $c^\star$; then query the decryption oracle on $(\!|C[c^\star]|\!)$, xor the result with Δ, and return 0 if it equals $(\!|m_0|\!)$ and 1 otherwise. An example of a scheme vulnerable to a universal attack of this form is the Zheng-Seberry cryptosystem [33], whose encryption algorithm is given by $e = \mathsf{f}(r) \,\|\, (G(r) \oplus (m \,\|\, H(m)))$; take $m_0 = 0^{|m|}$ and $\Delta = 1^{|m|}$.

5.2 Existential Attacks

To find existential attacks against a scheme with encryption algorithm e w.r.t. a set of assumptions Γ, we find universal attacks against modified versions of it. An example can elucidate better the point.

Consider the encryption algorithm of ZAEP [8], $e = \mathsf{f}(r \,\|\, G(r) \oplus m)$. To show that there is no blackbox reduction from the CPA security of ZAEP to the assumption $(\mathsf{f}, 0, |\mathsf{f}|)$, we instantiate f as follows

$$\mathsf{f}(a \,\|\, b) = a \,\|\, \mathsf{f}_2(b) \tag{1}$$

If f_2 is a one-way permutation, the permutation f satisfies the assumption $(\mathsf{f}, 0, |\mathsf{f}|)$. Using static distinguishability, we find an attack on $e' = r \,\|\, \mathsf{f}_2(G(r) \oplus m)$ given by the contexts

$$C = [*]_{|r|}^{|m|} \text{ and } C' = \mathsf{f}_2\left(G\left([*]_0^{|r|}\right)\right)$$

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

To show that there is no blackbox reduction of the CCA security of ZAEP to an arbitrary Γ, we use the instantiation

$$\mathsf{f}(a) = \mathsf{f}'\left([a]_0^{|a|-c}\right) \Big\| \, [a]_{|a|-c}^{c} \tag{2}$$

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 Γ (accounting for the size reduction by c), this instance of f satisfies Γ. 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 Γ. 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, to prove that there is no blackbox reduction for a fixed set of one-wayness assumptions Γ, we must either find a universal attack or instantiations for the trapdoor permutations that yield attacks and are compatible with *all* assumptions in Γ. For example, the instantiation (2) above is compatible with any set of assumptions, while (1) is compatible with all assumptions except those of the form (f, k, ℓ) with $0 < \ell$ and $k + \ell \leq |a|$. In addition to the aforementioned instantiations, we also use instantiations of the form

$$\mathsf{f}(a \,\|\, b \,\|\, c) = \mathsf{f}_1(a) \,\|\, b \oplus a \,\|\, c$$

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

6 Experimental Validation

We implemented the proof search and attack finding methods described in § 4 and § 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.

6.1 Security Analysis

To analyze the security of a scheme with encryption algorithm given by an expression $c^\star$ under a set of assumptions Γ, the toolset follows the workflow depicted in Figure 6.1:

1. Checks that $c^\star$ is well-typed and that encryption is invertible, i.e. $c^\star \vdash^\star m$;
2. Searches for attacks against CPA security;
3. Searches for proofs of CPA security. If a proof is found, computes the corresponding security bound;
4. Searches for malleability attacks against CCA security;
5. 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 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 [9, 16] 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.

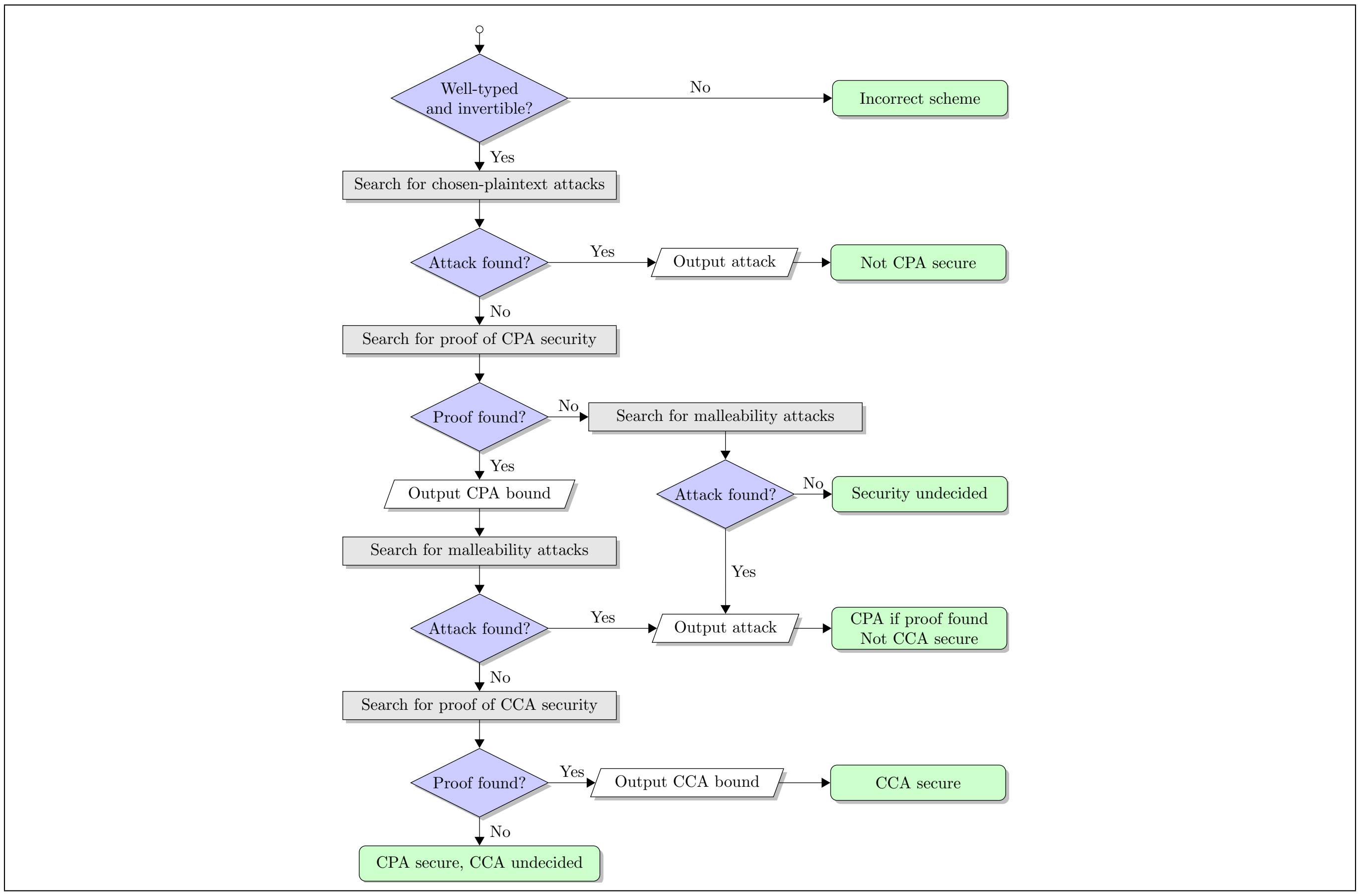
Well-typed and invertible?
No
Incorrect scheme
Yes
Search for chosen-plaintext attacks
Attack found?
Yes
Output attack
Not CPA secure
No
Search for proof of CPA security
Proof found?
No
Search for malleability attacks
Yes
Output CPA bound
Attack found?
No
Security undecided
Yes
Search for malleability attacks
Attack found?
Yes
Output attack
CPA if proof found Not CCA secure
No
Search for proof of CCA security
Proof found?
Yes
Output CCA bound
CCA secure
No
CPA secure, CCA undecided

Figure 9: Security analysis workflow

6.2 Practical Interpretation for RSA

Concrete security bounds can be used to guide the choice of practical parameters for realizations of encryption schemes based on RSA. In order to validate the quality of the bounds output by our tool, we implemented a method based on an extrapolation of the estimated cost of factoring the RSA-768 integer from the RSA Factoring Challenge [24], and on the lattice basis reduction of set partial-domain one-wayness to one-wayness for RSA [21].

Let t_N be the cost of factoring an N-bit RSA modulus: Kleinjung [24] estimates that $t_{768} = 2^{67}$, whereas Lenstra [26] suggests to extrapolate based on the ratio

$$\frac{t_N}{t_M} \approx \frac{\exp((1.9229)\log(N)^{1/3}\log(\log(N))^{2/3}}{\exp((1.9229)\log(M)^{1/3}\log(\log(M))^{2/3}}$$

The success probability of partially inverting RSA on its k most significant bits, $\mathbf{Succ}^{\mathsf{OW}}_{\mathsf{RSA}}(k,q,t)$ can be upper bounded by a function of $\mathbf{Succ}^{\mathsf{OW}}_{\mathsf{RSA}}(t')$, where t' depends on t.

Proposition 9 ([21])**.** *Let Θ be RSA with an ℓ-bit modulus such that $\ell < 2k$. Then,*

$$\mathbf{Succ}^{\mathsf{OW}}_{\Theta}(t)\left(\mathbf{Succ}^{\mathsf{OW}}_{\Theta}(k,q,t) - 2^{\ell-2k+6}\right) \le \mathbf{Succ}^{\mathsf{OW}}_{\Theta}(2t + q^2\ell^3)$$

Fixing a maximum number of queries to oracles and an admissible advantage p, our tool can estimate from the security bound obtained from a derivation, the minimum RSA modulus length N such that no adversary executing within time $t_{\mathcal{A}}$ achieves either a CPA or CCA advantage greater than p.

For instance, a reasonable security target might be that no CCA adversary executing within time 2^{128} and making at most 2^{60} hash queries and 2^{30} decryption queries achieves an advantage of more than 2^{-20}. From these values, the estimated cost of inverting RSA, and the bound found for the CCA security of OAEP under the same partial-domain one-wayness assumption as in [21], our tool estimates a minimum modulus length of 4864 bits, and a ciphertext overhead of around 200 bits.

6.3 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 $\mathsf{f}(x)$, $H(x)$, $x \oplus y$ 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 [28]. Concretely, we apply some simple filters during generation. For instance, given an expression e with holes, we check for the existence of a substitution σ for holes respecting the budget constraints such that $e\sigma \vdash^{\star} m$, 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.

6.4 Experiments

We evaluate our tools on encryption schemes generated under different budget constraints. Figure 1 summarizes the results of the automated security analysis of §6.1. In the figure, 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:

1. $\Gamma_1 = \{(\mathsf{f}, 0, |\mathsf{f}|) \mid \mathsf{f} \in \mathcal{F}(c^\star)\}$, i.e. assuming that all trapdoor permutations are one-way;
2. $\Gamma_2 = \{(\mathsf{f}, k_f, n_f) \mid \mathsf{f} \in \mathcal{F}(c^\star)\}$ such that $0 \leq k_f$ and $k_f + n_f \leq |\mathsf{f}|$ for all $\mathsf{f} \in \mathcal{F}(c^\star)$, i.e. one arbitrary one-wayness assumption for each trapdoor permutation;

The columns grouped under OW CPA report the results obtained when analyzing CPA security under Γ_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

Size	Total	OW CPA (% of Total)			CPA (% of Total)			CCA (% of CPA Proof + CPA Undec.)			
		Proof	Attack	Undec.	Proof	Attack	Undec.	Proof	Attack	NR	Undec.
4	2	1 (50.00%)	1 (50.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	2 (100.00%)	0 (0.00%)	0 (0.00%)
5	44	8 (18.18%)	36 (81.82%)	0 (0.00%)	12 (27.27%)	32 (72.73%)	0 (0.00%)	0 (0.00%)	13 (100.00%)	0 (0.00%)	0 (0.00%)
6	335	65 (19.40%)	270 (80.60%)	0 (0.00%)	93 (27.76%)	241 (71.94%)	1 (0.30%)	1 (0.98%)	96 (94.12%)	5 (4.90%)	0 (0.00%)
7	3263	510 (15.63%)	2735 (83.82%)	18 (0.55%)	750 (22.98%)	2475 (75.85%)	38 (1.16%)	45 (5.05%)	739 (82.94%)	45 (5.05%)	62 (6.96%)
8	32671	4430 (13.56%)	27894 (85.38%)	347 (1.06%)	6718 (20.56%)	25336 (77.55%)	617 (1.89%)	536 (6.26%)	6531 (76.25%)	306 (3.57%)	1192 (13.92%)
9	350111	43556 (12.44%)	301679 (86.17%)	4876 (1.39%)	66775 (19.07%)	274813 (78.49%)	8523 (2.43%)	7279 (8.16%)	62356 (69.93%)	3035 (3.40%)	16496 (18.50%)
10	644563	67863 (10.53%)	569314 (88.33%)	7386 (1.15%)	133476 (20.71%)	491189 (76.20%)	19898 (3.09%)	20140 (11.29%)	112993 (63.36%)	12794 (7.17%)	32397 (18.17%)
Total	1030989	116433 (11.29%)	901929 (87.48%)	12627 (1.22%)	207826 (20.16%)	794086 (77.02%)	29077 (2.82%)	28001 (10.11%)	182730 (65.95%)	16185 (5.84%)	50147 (18.10%)

Table 1: Evaluation of the tool on generated encryption schemes

results when analyzing CPA and CCA security under all assumptions of the form Γ_2. In this case, column Proof indicates the number of schemes proved secure under *some* assumption of the form Γ_2, column Attack the number of schemes for which an attack was found for *all* assumptions of the form Γ_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 [7, 25], but their proofs require random oracle *programmability*, which is out of the scope of our logics.

Validation 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 and Ohta [25]. 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 [25]. The results for CCA security are more nuanced: for about 20% of schemes, we fail to find proofs or attacks when they exist.

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 [Eqs] and [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.

7 Related Work

Our work lies at the intersection between symbolic and computational cryptography, and draws on verification techniques from both areas. We refer to [14, 18] for a recent account of symbolic verification and focus on verification tools and methods for the computational model, and cryptographic soundness.

Since its inception [2], cryptographic soundness was extended to many settings [18]. Despite this success, cryptographic soundness for constructions based on exclusive-or and one-way trapdoor permutations has remained elusive. Negative results such as [32] show that cryptographic soundness may indeed be very difficult to achieve for the setting of this paper.

CryptoVerif [13] and EasyCrypt [5] support the construction and verification of cryptographic proofs in the computational model. Both CryptoVerif and EasyCrypt provide a high level of automation, and CryptoVerif supports fully automated verification of many cryptographic protocols. In addition, EasyCrypt has been applied to verify security of several padding-based schemes. However, fully automated proofs of padding-based schemes are out of reach of these tools. [19] 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 [11] and REACT [27], 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.

In addition, a number of formalisms have been developed to reason about security of cryptographic primitives and constructions in the computational model [6, 23]. However, these systems reason about constructions described in mathematical vernacular, and are thus not readily amenable to automation. Similar formalisms exist for cryptographic protocols [29], but these are not automated either.

Finally, the batch mode of ZooCrypt is similar in spirit to the AVGI toolkit [28]. 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 whether generated protocols comply with the desired properties.

8 Conclusion

We have defined, implemented and evaluated proof systems to reason about security of public-key encryption schemes built from trapdoor permutations and hash functions. Our work sets a new methodology for analyzing cryptographic constructions in the computational model. Preliminary investigations suggest that our methodology extends well to other settings, and we intend to apply it for building a comprehensive database that includes classical cryptographic constructions such as digital signature schemes, modes of operation, and message authentication codes. We also plan to build on top of ZooCrypt a complete online interactive tutor that can generate exercises, help users finding an attack or a security proof, and check that a given solution is correct. Based on our experience with the interactive mode of ZooCrypt, we are convinced that, if well-executed, such a tutor could provide a very attractive vector for introducing students to cryptographic proofs!

We have also started to connect ZooCrypt with EasyCrypt, and implemented a prototype mechanism that translates successful derivations in our logics into EasyCrypt proofs—see the appendix for some more information. While generation of EasyCrypt proofs provides independent evidence of the soundness of the logics, our primary interest for this connection is to increase automation in EasyCrypt proofs. We expect that combining reflection and proof generation will deliver significant benefits for EasyCrypt. Reflection, a technique widely used in proof assistants would allow us to reify EasyCrypt proof goals into judgments of our logics that can be proved automatically by our tool; the initial proof goals could then be discharged replaying the script generated by the proof translation mechanism.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2001*, pages 104–115, New York, 2001. ACM.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.

[3] G. Bana and H. Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *1st Conference on Principles of Security and Trust – POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, pages 189–208, Heidelberg, 2012. Springer.

[4] G. Barthe, B. Grégoire, and S. Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101, New York, 2009. ACM.

[5] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg, 2011. Springer.

[6] G. Barthe, B. Grégoire, Y. Lakhnech, and S. Zanella-Béguelin. Beyond provable security. Verifiable IND-CCA security of OAEP. In *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196, Heidelberg, 2011. Springer.

[7] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella-Béguelin. Probabilistic relational reasoning for differential privacy. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*, pages 97–110, New York, 2012. ACM.

[8] G. Barthe, D. Pointcheval, and S. Zanella-Béguelin. Verified security of redundancy-free encryption from Rabin and RSA. In *19th ACM Conference on Computer and Communications Security, CCS 2012*, pages 724–735, New York, 2012. ACM.

[9] M. Baudet, V. Cortier, and S. Delaune. Yapa: A generic tool for computing intruder knowledge. In *Rewriting Techniques and Applications*, pages 148–163, Heidelberg, 2009. Springer.

[10] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. *Inf. Comput.*, 207(4):496–520, Apr. 2009. ISSN 0890-5401.

[11] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security, CCS 1993*, pages 62–73, New York, 1993. ACM.

[12] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology – EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111, Heidelberg, 1994. Springer.

[13] B. Blanchet. A computationally sound mechanized prover for security protocols. In *27th IEEE Symposium on Security and Privacy, S&P 2006*, pages 140–154. IEEE Computer Society, 2006.

[14] B. Blanchet. Security protocol verification: Symbolic and computational models. In *1st International Conference on Principles of Security and Trust, POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, pages 3–29, Heidelberg, 2012. Springer.

[15] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 275–291, Heidelberg, 2001. Springer.

[16] Ş. Ciobâcă, S. Delaune, and S. Kremer. Computing knowledge in security protocols under convergent equational theories. In *Automated Deduction–CADE-22*, pages 355–370. Springer, Heidelberg, 2009.

[17] J.-S. Coron, M. Joye, D. Naccache, and P. Paillier. Universal padding schemes for RSA. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 226–241, Heidelberg, 2002. Springer.

[18] V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[19] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *15th ACM Conference on Computer and Communications Security, CCS 2008*, pages 371–380, New York, 2008. ACM.

[20] D. Dolev and A. C.-C. Yao. On the security of public key protocols. In *22nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 1981*, pages 350–357, Heidelberg, 1981. IEEE Computer Society.

[21] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *J. Cryptology*, 17(2):81–104, 2004.

[22] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[23] R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. *J. Comput. Syst. Sci.*, 72(2):286–320, 2006.

[24] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350, Heidelberg, 2010. Springer.

[25] Y. Komano and K. Ohta. Taxonomical security consideration of OAEP variants. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A(5):1233–1245, 2006.

[26] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *J. Cryptology*, 14(4):255–293, 2001.

[27] T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Heidelberg, 2001. Springer.

[28] A. Perrig and D. Song. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *13th IEEE Workshop on Computer Security Foundations, CSFW 2000*, pages 64–76, Los Alamitos, 2000. IEEE Computer Society.

[29] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive trace properties for computational security. *Journal of Computer Security*, 18(6):1035–1073, 2010.

[30] V. Shoup. OAEP reconsidered. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259, Heidelberg, 2001. Springer.

[31] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[32] D. Unruh. The impossibility of computationally sound XOR. Cryptology ePrint Archive, Report 2010/389, 2010.

[33] Y. Zheng and J. Seberry. Practical approaches to attaining security against adaptively chosen ciphertext attacks. In *Advances in Cryptology – CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 292–304, Heidelberg, 1993. Springer.

A Definitions and Proofs

In this section we make precise some of the definitions presented informally in the body of the paper, and sketch proofs of the most important results.

A.1 Algebraic Expressions as Programs

We follow the approach of Barthe et al. [4] and represent algorithms and security experiments as pWHILE programs operating on bitstrings. A program is composed of a main command and a set of procedures built from the following grammar:

$$\begin{array}{rlll} \mathcal{C} & ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{deterministic assignment} \\ & \mid & \mathcal{V} \xleftarrow{\$} \{0,1\}^{\ell} & \text{uniform random sampling} \\ & \mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\ & \mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{loop} \\ & \mid & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E},\dots,\mathcal{E}) & \text{procedure call} \\ & \mid & \mathcal{C};\ \mathcal{C} & \text{sequence} \\ & \mid & \mathsf{return}\ \mathcal{E} & \text{return statement} \end{array}$$

Programs are strongly-typed; base types denote bitstrings of a fixed length and are of the form $\{0,1\}^{\ell}$; booleans are represented as single bits (i.e. bitstrings of length 1). A program denotes a function from an initial memory, which maps variables to bitstrings of the appropriate length, to a distribution over final memories; we refer the reader to [4] for a detailed description of the semantics of a more general probabilistic language. We denote $\Pr[c : E]$ the probability of event E w.r.t. the distribution obtained from running program c in an initial memory mapping variables to default values. We represent events as boolean expressions. In the remainder, we use foreach statements as syntactic sugar for while loops that iterate over lists or sets (in some arbitrary order).

We first define an interpretation of algebraic expressions as pWHILE programs. Let e be a well-typed algebraic expression. For each symbol of the form f or f^{-1} in e denoting a trapdoor permutation over $\{0,1\}^k$, we assume given a probabilistic key generation procedure $\mathcal{KG}_f$ and deterministic operators $f_{pk}, f_{sk}^{-1} : \{0,1\}^k \to \{0,1\}^k$.

Definition 10 (Semantics of algebraic expressions)**.** *Let e be a well-typed expression of type ℓ, and y a pWHILE variable of type $\{0,1\}^{\ell}$. Define the procedure $(\!|y \lhd e|\!)$ as follows:*

- *For each trapdoor permutation symbol f (resp. f^{-1}) in e, $(\!|y \lhd e|\!)$ takes as parameter a public key pk_f (resp. a secret key sk_f).*
- *For each variable $x \in \mathcal{V}(e)$, $(\!|y \lhd e|\!)$ takes as parameter a variable x of type $\{0,1\}^{|x|}$.*
- *For each hash function symbol $H \in \mathcal{H}(e)$ of domain type k and codomain type ℓ, $(\!|y \lhd e|\!)$ is parameterized by a procedure H of type $\{0,1\}^k \to \{0,1\}^{\ell}$.*
- *For each $r_i \in \mathcal{R}(e)$, let r_i be a fresh variable of type $\{0,1\}^{|r_i|}$. Define inductively the program*

$(\!|y \lhd e|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{x})$ as follows:

$$\begin{array}{lcl}
(\!|y \lhd x|\!) & \stackrel{\text{def}}{=} & y \leftarrow x \\
(\!|y \lhd r_i|\!) & \stackrel{\text{def}}{=} & y \leftarrow r_i \\
(\!|y \lhd 0^n|\!) & \stackrel{\text{def}}{=} & y \leftarrow 0^n \\
(\!|y \lhd \mathsf{f}(e)|\!) & \stackrel{\text{def}}{=} & (\!|y' \lhd e|\!);\ y \leftarrow f_{pk_f}(y') \\
(\!|y \lhd \mathsf{f}^{-1}(e)|\!) & \stackrel{\text{def}}{=} & (\!|y' \lhd e|\!);\ y \leftarrow f^{-1}_{sk_f}(y') \\
(\!|y \lhd H(e)|\!) & \stackrel{\text{def}}{=} & (\!|y' \lhd e|\!);\ y \leftarrow H(y') \\
(\!|y \lhd e_1 \oplus e_2|\!) & \stackrel{\text{def}}{=} & (\!|y_1 \lhd e_1|\!);\ (\!|y_2 \lhd e_2|\!);\ y \leftarrow y_1 \oplus y_2 \\
(\!|y \lhd e_1 \,\|\, e_2|\!) & \stackrel{\text{def}}{=} & (\!|y_1 \lhd e_1|\!);\ (\!|y_2 \lhd e_2|\!);\ y \leftarrow y_1 \,\|\, y_2 \\
(\!|y \lhd [e]^{\ell}_{n}|\!) & \stackrel{\text{def}}{=} & (\!|y' \lhd e|\!);\ y \leftarrow [y']^{\ell}_{n}
\end{array}$$

where y', y_1 and y_2 are fresh variables.

Finally, define

$$(\!|e|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{x}) \stackrel{\text{def}}{=} \mathsf{foreach}\ r_i \in \mathcal{R}(e)\ \mathsf{do}\ r_i \stackrel{\$}{\leftarrow} \{0,1\}^{|r_i|}\ \mathsf{end};\ (\!|y \lhd e|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{x});\ \mathsf{return}\ y$$

Our proofs are in the random oracle model; we represent hash functions as pWhile procedures implemented in two equivalent ways: either as *lazy* or as *eager* oracles. A lazy implementation of a random oracle $H : \{0,1\}^k \to \{0,1\}^\ell$ is a stateful procedure that answers queries by sampling answers uniformly on demand and maintaining a map $\boldsymbol{H}$ of previous queries and answers:

```
Procedure $H(x) \stackrel{\text{def}}{=}$
  if $x \notin \mathsf{dom}(\boldsymbol{H})$ then $\boldsymbol{H}[x] \stackrel{\$}{\leftarrow} \{0,1\}^\ell$;
  return $\boldsymbol{H}[x]$
```

If the map $\boldsymbol{H}$ is not accessed elsewhere, pre-sampling the answers to some queries at the beginning of a game does not make any difference. An *eager* implementation fully populates $\boldsymbol{H}$ with uniform and independently sampled values, converting H into a side-effect free procedure:

$$\textbf{Procedure } H(x) \stackrel{\text{def}}{=} \mathsf{return}\ \boldsymbol{H}[x]$$

Proposition 11 (Eager/lazy sampling equivalence)**.** *Let c be a pWhile program implementing a procedure $H : \{0,1\}^k \to \{0,1\}^\ell$ as a lazy random oracle and c' be identical to c, but implementing H as an eager random oracle. If variable $\boldsymbol{H}$ does not occur in c (and c') except in the definition of H, then for any event ϕ where $\boldsymbol{H}$ does not occur free,*

$$\Pr[\boldsymbol{H} \leftarrow \emptyset;\ c : \phi] = \Pr\left[\mathsf{init}_H;\ c' : \phi\right]$$

where $\mathsf{init}_H \stackrel{\text{def}}{=} \mathsf{foreach}\ x \in \{0,1\}^k\ \mathsf{do}\ \boldsymbol{H}[x] \stackrel{\$}{\leftarrow} \{0,1\}^\ell$.

When all procedures $\vec{H}$ are implemented as eager random oracles, $(\!|e|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{x})$ is side-effect free. We give an alternative side-effect free semantics to algebraic expressions as pWhile expressions.

Definition 12 (Alternative semantics of expressions)**.** *Let e be a well-typed algebraic expression. Define the pWhile probabilistic expression $\langle\!|e|\!\rangle$ as follows:*

- *For each trapdoor permutation symbol* f *(resp.* f^{-1}*) in* e, $\langle\!\lbrack e \rbrack\!\rangle$ *takes as parameter a public key* pk_f *(resp. a secret key* sk_f*).*
- *For each variable* $x \in \mathcal{V}(e)$, $\langle\!\lbrack e \rbrack\!\rangle$ *takes as parameter a variable* x *of type* $\{0,1\}^{|x|}$.
- *For each hash function symbol* $H \in \mathcal{H}(e)$ *of domain type* k *and codomain type* ℓ, $\langle\!\lbrack e \rbrack\!\rangle$ *is parameterized by a variable* $\boldsymbol{H}$ *denoting a finite map from* $\{0,1\}^k$ *to* $\{0,1\}^\ell$.
- *For each* $r_i \in \mathcal{R}(e)$, *let* r_i *be a fresh variable of type* $\{0,1\}^{|r_i|}$. *Define inductively the program* $\langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})$ *as follows:*

$$\begin{array}{lcl}
\langle\!\lbrack x \rbrack\!\rangle & \stackrel{\text{def}}{=} & x \\
\langle\!\lbrack r_i \rbrack\!\rangle & \stackrel{\text{def}}{=} & r_i \\
\langle\!\lbrack 0^n \rbrack\!\rangle & \stackrel{\text{def}}{=} & 0^n \\
\langle\!\lbrack \mathsf{f}(e) \rbrack\!\rangle & \stackrel{\text{def}}{=} & f_{pk_f}(\langle\!\lbrack e \rbrack\!\rangle) \\
\langle\!\lbrack \mathsf{f}^{-1}(e) \rbrack\!\rangle & \stackrel{\text{def}}{=} & f^{-1}_{sk_f}(\langle\!\lbrack e \rbrack\!\rangle) \\
\langle\!\lbrack H(e) \rbrack\!\rangle & \stackrel{\text{def}}{=} & \boldsymbol{H}[\langle\!\lbrack e \rbrack\!\rangle'] \\
\langle\!\lbrack e_1 \oplus e_2 \rbrack\!\rangle & \stackrel{\text{def}}{=} & \langle\!\lbrack e_1 \rbrack\!\rangle \oplus \langle\!\lbrack e_2 \rbrack\!\rangle \\
\langle\!\lbrack e_1 \,\|\, e_2 \rbrack\!\rangle & \stackrel{\text{def}}{=} & \langle\!\lbrack e_1 \rbrack\!\rangle \,\|\, \langle\!\lbrack e_2 \rbrack\!\rangle \\
\langle\!\lbrack [e]_n^\ell \rbrack\!\rangle & \stackrel{\text{def}}{=} & [\langle\!\lbrack e \rbrack\!\rangle]_n^\ell
\end{array}$$

Finally, define $[\![e]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) = \mathsf{let}\ r_1 \xleftarrow{\$} \{0,1\}^{|r_1|}, \dots, r_n \xleftarrow{\$} \{0,1\}^{|r_n|}\ \mathsf{in}\ \langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})$.

In the remainder, use the notational convention that for an expression e, the functions $\langle\!\lbrack e \rbrack\!\rangle$, $[\![e]\!]$ and $(\!|e|\!)$ ignore additional arguments for variables that do not occur in e.

Lemma 13 (Substitution). *For any well-typed pair of algebraic expressions* e, e', *and* $r \in \mathcal{R}$, $x \in \mathcal{X}$,

$$\langle\!\lbrack e\,\{e'/r\} \rbrack\!\rangle = \mathsf{let}\ r = \langle\!\lbrack e' \rbrack\!\rangle\ \mathsf{in}\ \langle\!\lbrack e \rbrack\!\rangle \qquad \langle\!\lbrack e\,\{e'/x\} \rbrack\!\rangle = \mathsf{let}\ x = \langle\!\lbrack e' \rbrack\!\rangle\ \mathsf{in}\ \langle\!\lbrack e \rbrack\!\rangle$$

Proof. Immediate by structural induction over e. □

We have the following correspondence between the two interpretations of algebraic expressions.

Lemma 14 (Correspondence). *Let* e *be an algebraic expression of type* ℓ, *and suppose all procedures* $\vec{H}$ *in* $\mathcal{H}(e)$ *are implemented as eager random oracles using maps* $\vec{\boldsymbol{H}}$. *Then, for all* $\vec{pk}$, $\vec{sk}$, $\vec{x}$, *and* $v \in \{0,1\}^\ell$,

$$\Pr\left[y \leftarrow (\!|e|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right] = \Pr\left[y \leftarrow [\![e]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right]$$

Proof. By induction in the structure of e, inlining calls to hash oracles in $(\!|e|\!)$. □

We provide an interpretation of equational reasoning in terms of the side-effect free semantics of algebraic expressions.

Lemma 15 (Soundness of equational reasoning). *Let* e_0, e_1 *be well-typed algebraic expressions of type* ℓ *such that* $e_0 =_E e_1$. *For any trapdoor permutation symbol* $\mathsf{f} \in \mathcal{F}(e_0, e_1)$, *let* (pk_f, sk_f) *be a pair of keys generated using* $\mathcal{KG}_f$; *denote* $\vec{pk}$ *(resp.* $\vec{sk}$*) the vector of all such keys. For any* $\vec{\boldsymbol{H}}$ *and* $\vec{x}$,

$$[\![e_0]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) = [\![e_1]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})$$

i.e., both expressions generate the same distribution on $\{0,1\}^\ell$.

Proof. The proof follows by induction on the derivation of $e_0 =_E e_1$, the interpretation of expressions as side-effect free pWHILE expressions and the properties of bitstring operators at the pWHILE language level. It suffices to show that for all axioms $e = e'$ in Figure 1 and values for variables corresponding to random bitstrings, the bitstrings corresponding to $\langle\!\langle e \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})$ and $\langle\!\langle e \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})$ coincide, and that this equality is a congruence. $\square$

A.2 Soundness of Deducibility and Apartness

We provide computational interpretations of deducibility and apartness in terms of the side-effect free interpretation of algebraic expressions.

Lemma 16 (Soundness of deducibility)**.** *Let e_0, e_1 be well-typed algebraic expressions such that $e_0 \vdash e_1$. There exists a pWHILE procedure P such that for any appropriately generated $\vec{pk}$, $\vec{sk}$, and for any $\vec{\boldsymbol{H}}$, $\vec{x}$, and $v \in \{0,1\}^{|e_1|}$,*

$$\Pr\left[y \leftarrow P^{\vec{H}}\left(\vec{pk}, [\![e_0]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})\right) : y = v\right] = \Pr\left[y \leftarrow [\![e_1]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right]$$

where procedures $\vec{H}$ are implemented as eager random oracles using maps $\vec{\boldsymbol{H}}$.

Proof. Let C be the context witnessing the deducibility $e_0 \vdash e_1$. From Definition 5, $C[e_0]$ is well-typed, no symbol f^{-1} occurs in C, $\mathcal{R}(C) = \emptyset$, $\mathcal{X}(C) = \{*\}$, and $C[e_0] =_E e_1$. Define $P \stackrel{\text{def}}{=} (\!|C|\!)$. P has the right type: it expects as parameters a vector of public keys and a single value of type $\{0,1\}^{|e_0|}$ for variable $*$, and returns a value in $\{0,1\}^{|e_1|}$.

From Lemma 14,

$$\Pr\left[y \leftarrow P^{\vec{H}}\left(\vec{pk}, [\![e_0]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})\right) : y = v\right] = \Pr\left[y \leftarrow [\![C]\!]^{\vec{\boldsymbol{H}}}\left(\vec{pk}, [\![e_0]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})\right) : y = v\right]$$

Observe that since $\mathcal{R}(C) = \emptyset$, $[\![C]\!]$ is deterministic, and its semantics can be safely composed with $[\![e_0]\!]$,

$$\Pr\left[y \leftarrow [\![C]\!]^{\vec{\boldsymbol{H}}}\left(\vec{pk}, [\![e_0]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})\right) : y = v\right] = \Pr\left[y \leftarrow [\![C[e_0]]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right]$$

Finally, from Lemma 15 and $C[e_0] =_E e_1$,

$$\Pr\left[y \leftarrow [\![C[e_0]]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right] = \Pr\left[y \leftarrow [\![e_1]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right] \quad \square$$

Proposition 17. *Let e_0, e_1 be well-typed algebraic expressions such that $e_0 \vdash^{\star} e_1$. There exists a pWHILE procedure P such that for any appropriately generated $\vec{pk}$, $\vec{sk}$, and for any $\vec{\boldsymbol{H}}$, $\vec{x}$, and $v \in \{0,1\}^{|e_1|}$,*

$$\Pr\left[y \leftarrow P^{\vec{H}}(\vec{pk}, \vec{sk}, [\![e_0]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x})) : y = v\right] = \Pr\left[y \leftarrow [\![e_1]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : y = v\right]$$

Proof. Analogous to previous lemma, except that the context witnessing the deducibility may contain occurrences of f^{-1}, and thus the procedure P has to take the corresponding secret keys as additional parameters. $\square$

Lemma 18 (Soundness of apartness)**.** *Let s, t be well-typed algebraic expressions of type ℓ such that $s \sharp t$ can be derived from the rules of Figure 2 under axioms in Γ. For any appropriately generated $\vec{pk}$, $\vec{sk}$, uniformly sampled $\vec{\boldsymbol{H}}$, and any $\vec{x}$ satisfying Γ,*

$$\Pr\left[v \leftarrow [\![s \,\|\, t]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : [v]_0^\ell = [v]_\ell^\ell\right]$$

is a negligible function of the security parameter (recall that types are interpreted as non-constant polynomials in the security parameter).

Proof. By structural induction on the derivation of $s \sharp t$.

Case [Ax]**:** Follows from the assumption that $\vec{x}$ satisfies Γ.

Case [Ctxt]**:** By contradiction. Assume that *not* $s \sharp t$, i.e. the values of s and t coincide with non-negligible probability. This contradicts the hypothesis $C[s] \sharp C[s]$, since it implies that the values of $C[s]$ and $C[t]$ coincide with non-negligible probability.

Case [Rnd]**:** Since $s \notin \mathcal{R}(s)$, the value $[v]_0^\ell$ of the random bitstring r is distributed uniformly and independent of $[v]_\ell^\ell$. Thus, the probability that $[v]_0^\ell$ and $[v]_\ell^\ell$ coincide is $2^{-|r|}$. This probability is negligible because $|r|$ is interpreted as a non-constant polynomial in the security parameter.

Case [EqE]**:** From Lemma 15, since $t =_E u$,

$$\Pr\left[v \leftarrow [\![s \,\|\, t]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : [v]_0^\ell = [v]_\ell^\ell\right] = \Pr\left[v \leftarrow [\![s \,\|\, u]\!]^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{x}) : [v]_0^\ell = [v]_\ell^\ell\right]$$

which is negligible from the inductive hypothesis.

Case [Hash]**:** The map $\boldsymbol{H}$ is uniformly sampled, and $\boldsymbol{H}[e]$ is independent of $(\oplus_{i=1}^k \boldsymbol{H}[e_i]) \oplus s$ conditioned on the value of e being different from the values of $e_1, \ldots, e_k$, which occurs with overwhelming probability. The same argument used in the the [Rnd] case applies. □

Note that rules are incomplete (e.g. one cannot prove $r \sharp \mathsf{f}(r)$), but suffice for the use we made of them for finding attacks.

A.3 Soundness of CPA logic

In this section, all definitions and proofs are relative to fixed finite sets $\mathcal{F}$ and $\mathcal{H}$ of (typed) trapdoor permutation and hash function symbols. We assume that all expressions and events under consideration only contain trapdoor permutation and hash function symbols in these sets.

Definition 19 (CPA experiment)**.** *Let e be a well-typed algebraic expression such that $\mathcal{X}(e) \subseteq \{m\}$. Let $\mathcal{R}$ be a set of random bitstrings symbols such that $\mathcal{R}(e) \subseteq \mathcal{R}$. Given a* CPA *adversary $\mathcal{A}$ and $\vec{q} \in \mathbb{N}^{|\mathcal{H}|}$, define the game* CPA$(e, \mathcal{R}, \vec{q}, \mathcal{A})$ *as follows*

```
Game CPA$(e, \mathcal{R}, \vec{q}, \mathcal{A}) \stackrel{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow$ nil; init$_H$;
  foreach $r \in \mathcal{R}$ do $r \stackrel{\$}{\leftarrow} \{0,1\}^{|r|}$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk})$;
  $b \stackrel{\$}{\leftarrow} \{0,1\}$;
  $c^\star \leftarrow (\!|e|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_\mathcal{A}}(c^\star, \sigma)$

Procedure $H_\mathcal{A}(x) \stackrel{\text{def}}{=}$
  if $|\boldsymbol{L}_H| < q_H$ then $\boldsymbol{L}_H \leftarrow x :: \boldsymbol{L}_H$; return $H(x)$
  else return $\perp$

Procedure $H(x) \stackrel{\text{def}}{=}$ return $\boldsymbol{H}[x]$
```

where init_H *is defined as above. The adversary* $\mathcal{A}$ *is represented as a pair of procedures* $(\mathcal{A}_1, \mathcal{A}_2)$ *that are given access to a hash oracle* H *through a wrapper* $H_{\mathcal{A}}$ *that stores queries in a list* $\boldsymbol{L}_H$ *and limits the number of distinct queries to* q_H*.*

We let $\mathsf{CPA}_{\mathcal{G}}(e, \mathcal{R}, \vec{q}, \mathcal{A})$ *denote the same experiment except that the hash oracles* $\mathcal{G} \subseteq \mathcal{H}$ *are implemented lazily and that* $c^\star$ *is computed using* $(\!|\cdot|\!)$ *instead of* $\langle\!|\cdot|\!\rangle$*:*

```
Game $\mathsf{CPA}_{\mathcal{G}}(e, \mathcal{R}, \vec{q}, \mathcal{A}) \stackrel{\text{def}}{=}$
    foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
    foreach $H \in \mathcal{G}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\boldsymbol{H} \leftarrow \emptyset$;
    foreach $H \in \mathcal{H} \setminus \mathcal{G}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
    foreach $r \in \mathcal{R}$ do $r \stackrel{\$}{\leftarrow} \{0,1\}^{|r|}$;
    $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_{\mathcal{A}}}(\vec{pk})$;
    $b \stackrel{\$}{\leftarrow} \{0,1\}$;
    $(\!|c^\star \lhd e|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, m_b)$;
    $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_{\mathcal{A}}}(c^\star, \sigma)$
```

Definition 20 (Probability of events)**.** *We interpret the event* Guess *as* $b = \bar{b}$ *and an event* $\mathsf{Ask}(H, e)$ *as* $e \in \boldsymbol{L}_H$*. Under this interpretation, we naturally extend the semantic function* $\langle\!|\cdot|\!\rangle$ *from expressions to events. Let* ϕ *be an event of the* CPA *logic. For all games* G *such that all hash oracles used in* ϕ *are implemented eagerly and all arguments required to interpret* ϕ *are defined,*

$$\Pr\left[\!\left[G : \phi\right]\!\right] \stackrel{\text{def}}{=} \Pr\left[G : \langle\!|\phi|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, b, \bar{b}, m_b, \boldsymbol{L}_{\vec{H}})\right].$$

Lemma 21. *Let* e *be a well-typed algebraic expression such that* $\mathcal{X}(e) \subseteq \{m\}$*. Let* ϕ *be a* CPA *event such that no expression with head symbol* $H \in \mathcal{G}$ *occurs in* ϕ*. Let* $\mathcal{R}$ *be a set of random bitstrings symbols such that* $\mathcal{R}(e, \phi) \subseteq \mathcal{R}$*. For any* CPA *adversary* $\mathcal{A}$ *and* $\vec{q} \in \mathbb{N}^{|\mathcal{H}|}$*,*

$$\Pr\left[\!\left[\mathsf{CPA}(e, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] = \Pr\left[\!\left[\mathsf{CPA}_{\mathcal{G}}(e, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right]$$

Proof. First apply Lemma 14 to switch to an interpretation of expressions and events as programs. For events, append the corresponding procedure calls to the experiments, e.g.

$$\Pr\left[\!\left[\mathsf{CPA}(e, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] = \Pr\left[\mathsf{CPA}(e, \mathcal{R}, \vec{q}, \mathcal{A});\ (\!|\phi \lhd \phi|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, b, \bar{b}, m_b, \boldsymbol{L}_{\vec{H}}) : \phi\right]$$

This removes any occurrence of maps $\boldsymbol{H}$ from the games except in the definition of oracle H. Then, repeatedly apply Proposition 11 to conclude. □

Next, we prove the soundness of the CPA logic.

Lemma 22. *Let* $c^\star$ *be a well-typed algebraic expression with* $\mathcal{X}(c^\star) \subseteq \{m\}$*,* ϕ *an event of the* CPA *logic, and* $\hat{p}$ *a probability tag. Let* ∇ *be a derivation of* $\vDash_{\hat{p}} c^\star : \phi$ *from the rules of Figure 3. For any set of random bitstrings* $\mathcal{R}$ *such that* $\mathcal{R}(c^\star, \phi) \subseteq \mathcal{R}$*, any* CPA *adversary* $\mathcal{A}$*, and* $\vec{q} \in \mathbb{N}^{|\mathcal{H}|}$*,*

$$\Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \leq \mathcal{B}_{(t_A, \vec{q})}(\nabla)$$

Proof. By structural induction on the derivation ∇.

Case [Opt]**:** It suffices to prove that for any expression e and random bitstring r such that $r \notin \mathcal{R}(e)$,

$$\Pr \llbracket \mathsf{CPA}(c^\star \{e \oplus r/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \{e \oplus r/r\} \rrbracket \leq \Pr \llbracket \mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \rrbracket$$

From Lemma 13, and because $H_\mathcal{A}$ does not depend on r, the experiment $\mathsf{CPA}(c^\star \{e \oplus r/r\}, \mathcal{R}, \vec{q}, \mathcal{A})$ is equivalent to

```
Game $G \stackrel{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
  foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk})$;
  $b \xleftarrow{\$} \{0,1\}$;
  $r \leftarrow \llparenthesis e \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b) \oplus r$;
  $c^\star \leftarrow \llparenthesis c^\star \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_\mathcal{A}}(c^\star, \sigma)$
```

and

$$\Pr \llbracket \mathsf{CPA}(c^\star \{e \oplus r/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \{e \oplus r/r\} \rrbracket = \Pr \llbracket G : \phi \rrbracket$$

Moreover, since $r \notin \mathcal{R}(e)$, for all $v \in \{0,1\}^{|r|}$,

$$\Pr \left[r \xleftarrow{\$} \{0,1\}^{|r|} : \llparenthesis e \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b) \oplus r = v \right] = \Pr \left[r \xleftarrow{\$} \{0,1\}^{|r|} : r = v \oplus \llparenthesis e \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b) \right] = 2^{-|r|}$$

This means that the value of r in G used to compute $c^\star$ and subsequently is uniformly distributed and independent of $\vec{pk}, \vec{sk}, \vec{\boldsymbol{H}}, \mathcal{R} \setminus \{r\}$, just as in the experiment $\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A})$, and

$$\Pr \llbracket G : \phi \rrbracket = \Pr \llbracket \mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \rrbracket$$

Case [Perm]**:** It suffices to prove that for any expression e, permutation f and random bitstring r,

$$\Pr \llbracket \mathsf{CPA}(c^\star \{\mathsf{f}(r)/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \{\mathsf{f}(r)/r\} \rrbracket \leq \Pr \llbracket \mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \rrbracket$$

From Lemma 13, the experiment $\mathsf{CPA}(c^\star \{\mathsf{f}(r)/r\}, \mathcal{R}, \vec{q}, \mathcal{A})$ is equivalent to

```
Game $G \stackrel{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
  foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $r \leftarrow f_{pk_f}(r)$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk})$;
  $b \xleftarrow{\$} \{0,1\}$;
  $c^\star \leftarrow \llparenthesis c^\star \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_\mathcal{A}}(c^\star, \sigma)$
```

and

$$\Pr \llbracket \mathsf{CPA}(c^\star \{\mathsf{f}(r)/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi \{\mathsf{f}(r)/r\} \rrbracket = \Pr \llbracket G : \phi \rrbracket$$

Moreover, since f_{pk_f} is a bijection on $\{0,1\}^{|r|}$ (with inverse $f^{-1}_{sk_f}$), we have that for all $v \in \{0,1\}^{|r|}$,

$$\Pr\left[r \xleftarrow{\$} \{0,1\}^{|r|} : f_{pk_f}(r) = v\right] = \Pr\left[r \xleftarrow{\$} \{0,1\}^{|r|} : r = f^{-1}_{sk_f}(v)\right] = 2^{-|r|}.$$

This means that the value of r in G after the instruction $r \leftarrow f_{pk_f}(r)$ is uniformly distributed and independent of $\vec{pk}, \vec{sk}, \vec{\boldsymbol{H}}, \mathcal{R} \setminus \{r\}$, just as in the experiment $\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A})$, and

$$\Pr\left[\!\left[G : \phi\right]\!\right] = \Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right]$$

Case [Merge] **and** [Split]**:** It suffices to show that for any random bitstring symbols r, r_1, r_2 such that $r_1, r_2 \notin \mathcal{R}(c^\star, \phi)$ and $r_1 \neq r_2$,

$$\Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] = \Pr\left[\!\left[\mathsf{CPA}(c^\star\left\{r_1 \,\|\, r_2/r\right\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\left\{r_1 \,\|\, r_2/r\right\}\right]\!\right]$$

From Lemma 13, the experiment $\mathsf{CPA}(c^\star\left\{r_1 \,\|\, r_2/r\right\}, \mathcal{R}, \vec{q}, \mathcal{A})$ is equivalent to

```
Game $G \stackrel{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
  foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $r_1 \xleftarrow{\$} \{0,1\}^{|r_1|}$; $r_2 \xleftarrow{\$} \{0,1\}^{|r_2|}$
  $r \leftarrow r_1 \,\|\, r_2$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk})$;
  $b \xleftarrow{\$} \{0,1\}$;
  $c^\star \leftarrow (\!|c^\star|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_\mathcal{A}}(c^\star, \sigma)$
```

where $|r| = |r_1| + |r_2|$. Note that, for all $v \in \{0,1\}^{|r_1|+|r_2|}$ we have that

$$\Pr\left[r \xleftarrow{\$} \{0,1\}^{|r|} : r = v\right] = \Pr\left[r_1 \xleftarrow{\$} \{0,1\}^{|r_1|}; r_2 \xleftarrow{\$} \{0,1\}^{|r_2|}; r \leftarrow r_1 \,\|\, r_2 : r = v\right] = 2^{-(|r_1|+|r_2|)} = 2^{-|r|}.$$

This means that the value of r in G after the instruction $r \leftarrow r_1 \,\|\, r_2$ is uniformly distributed and independent of $\vec{pk}, \vec{sk}, \vec{\boldsymbol{H}}, \mathcal{R} \setminus \{r, r_1, r_2\}$. Since $r_1, r_2 \notin \mathcal{R}(c^\star, \phi)$, the distribution of $c^\star$ is also the same in both experiments, and

$$\Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] = \Pr\left[\!\left[\mathsf{CPA}(c^\star\left\{r_1 \,\|\, r_2/r\right\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\left\{r_1 \,\|\, r_2/r\right\}\right]\!\right]$$

Case [Sub]**:** It suffices to prove that for any expressions $c_1^\star, c_2^\star$ and events ϕ_1, ϕ_2 such that $c_1^\star =_E c_2^\star$ and $\phi_1 \Longrightarrow_E \phi_2$,

$$\Pr\left[\!\left[\mathsf{CPA}(c_1^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi_1\right]\!\right] \leq \Pr\left[\!\left[\mathsf{CPA}(c_2^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi_2\right]\!\right]$$

From Lemma 15, the two experiments above are equivalent and event ϕ_2 happens whenever ϕ_1 happens, and thus the inequality holds.

Case [Fail$_1$] **and** [Fail$_2$]**:** Let $e, c^\star$ be well-typed expressions, r a random bitstring, and H a hash function symbol such that $r \notin \mathcal{R}(e)$ and no expression with head symbol H occurs in $c^\star$, ϕ, or e. To prove the first case, it suffices to prove

$$\Pr\left[\mathsf{CPA}(c^\star\left\{H(e)/r\right\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\left\{H(e)/r\right\}\right] \leq \Pr\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right] + \Pr\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Ask}(H, e)\right]$$

We first use Lemma 21 to switch to a lazy implementation of H in both experiments. We then introduce two intermediate games G_1 and G_2, that are semantically equivalent to games $\mathsf{CPA}_{\{H\}}(c^\star\{H(e)/r\}, \mathcal{R}, \vec{q}, \mathcal{A})$ and $\mathsf{CPA}_{\{H\}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A})$, respectively. The only difference is that their code is instrumented to set a flag **bad** to true whenever the failure event $e \in \boldsymbol{L}_H$ is triggered. Moreover, these two games are syntactically identical except after program points raising the **bad** flag.

```
Game $G_1 \stackrel{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H} \setminus \{H\}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
  foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\boldsymbol{H} \leftarrow \emptyset$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk})$;
  $b \xleftarrow{\$} \{0,1\}$;
  $(\!|v \lhd e|\!)(\vec{pk}, \vec{sk}, m_b)$;
  if $v \in \boldsymbol{L}_H$ then $\mathbf{bad} \leftarrow \mathsf{true}$; $r \leftarrow \boldsymbol{H}[v]$;
  else $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $(\!|c^\star \lhd c^\star|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H'}_\mathcal{A}}(c^\star, \sigma)$
Procedure $H'_\mathcal{A}(x) \stackrel{\text{def}}{=}$
  if $|\boldsymbol{L}_H| < q_H$ then
    $\boldsymbol{L}_H \leftarrow x :: \boldsymbol{L}_H$;
    if $x = v$ then $\mathbf{bad} \leftarrow \mathsf{true}$; return $r$
    else return $H(x)$
  else return $\perp$
```

```
Game $G_2 \stackrel{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H} \setminus \{H\}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
  foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\boldsymbol{H} \leftarrow \emptyset$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk})$;
  $b \xleftarrow{\$} \{0,1\}$;
  $(\!|v \lhd e|\!)(\vec{pk}, \vec{sk}, m_b)$;
  if $v \in \boldsymbol{L}_H$ then $\mathbf{bad} \leftarrow \mathsf{true}$; $r \xleftarrow{\$} \{0,1\}^{|r|}$
  else $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $(\!|c^\star \lhd c^\star|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H'}_\mathcal{A}}(c^\star, \sigma)$
Procedure $H'_\mathcal{A}(x) \stackrel{\text{def}}{=}$
  if $|\boldsymbol{L}_H| < q_H$ then
    $\boldsymbol{L}_H \leftarrow x :: \boldsymbol{L}_H$;
    if $x = v$ then $\mathbf{bad} \leftarrow \mathsf{true}$; return $H(x)$
    else return $H(x)$
  else return $\perp$
```

To see why experiment G_1 is equivalent to $\mathsf{CPA}_{\{H\}}(c^\star\{H(e)/r\}, \mathcal{R}, \vec{q}, \mathcal{A})$ (and $\mathsf{CPA}(c^\star\{H(e)/r\}, \mathcal{R}, \vec{q}, \mathcal{A})$) observe that in the latter r is never used after it is initialized. This is because $H_\mathcal{A}$ does not depend on r and r is substituted for $H(e)$ in $c^\star$ and ϕ. Moreover, from Lemma 13, the instruction

$$c^\star \leftarrow (\!|c^\star\{H(e)/r\}|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$$

can be written equivalently as

$$r \leftarrow (\!|H(e)|\!);\ c^\star \leftarrow (\!|c^\star|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$$

Because $r \notin \mathcal{R}(e)$ and $H \notin \mathcal{H}(e)$, this is in turn equivalent to

$$v \leftarrow (\!|e|\!)(\vec{pk}, \vec{sk}, m_b);\ r \leftarrow \boldsymbol{H}[v];\ c^\star \leftarrow (\!|c^\star|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$$

Since G_1 and G_2 are syntactically identical *up to* **bad** and **bad** $\iff v \in \boldsymbol{L}_H$,

1. $\Pr[G_1 : v \in \boldsymbol{L}_H] = \Pr[G_2 : v \in \boldsymbol{L}_H]$
2. for any event ϕ, $\Pr[G_1 : \phi \wedge v \notin \boldsymbol{L}_H] = \Pr[G_2 : \phi \wedge v \notin \boldsymbol{L}_H]$

From Shoup's Fundamental Lemma [31],

$$\begin{aligned}
&\Pr[\![\mathsf{CPA}(c^\star\{H(e)/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\{H(e)/r\}]\!] \\
&\quad = \Pr[\![G_1 : \phi]\!] \\
&\quad \le \Pr[\![G_2 : \phi]\!] + \Pr[G_2 : v \in \boldsymbol{L}_H] \\
&\quad = \Pr[\![\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi]\!] + \Pr[\![\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Ask}(H, e)]\!]
\end{aligned}$$

The soundness of rule [Fail_2] also follows from the above:

$$\begin{aligned}
&\Pr\left[\!\left[\mathsf{CPA}(c^\star\{H(e)/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\{H(e)/r\}\right]\!\right] \\
&\quad = \Pr\left[\!\left[G_1 : \phi\right]\!\right] \\
&\quad = \Pr\left[G_1 : (\!|\phi|\!) \wedge v \in \boldsymbol{L}_H\right] + \Pr\left[G_1 : (\!|\phi|\!) \wedge v \notin \boldsymbol{L}_H\right] \\
&\quad = \Pr\left[G_1 : (\!|\phi|\!) \wedge v \in \boldsymbol{L}_H\right] + \Pr\left[G_2 : (\!|\phi|\!) \wedge v \notin \boldsymbol{L}_H\right] \\
&\quad \le \Pr\left[\!\left[G_1 : \phi \wedge \mathsf{Ask}(H, v)\right]\!\right] + \Pr\left[\!\left[G_2 : \phi\right]\!\right] \\
&\quad = \Pr\left[\!\left[\mathsf{CPA}(c^\star\{H(e)/r\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\{H(e)/r\} \wedge \mathsf{Ask}(H, e)\right]\!\right] + \Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \quad \square
\end{aligned}$$

Case [Ind]**:** Since $\mathcal{X}(c^\star) = \emptyset$, the value of $c^\star$ in the experiment in the conclusion does not depend on the variable m_b. Because the behavior of $H_{\mathcal{A}}$ is also independent of the challenge bit b, we can defer sampling b until the end of the experiment:

```
Game $G \overset{\text{def}}{=}$
  foreach $f \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow$ nil; $\mathsf{init}_H$;
  foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_{\mathcal{A}}}(\vec{pk})$;
  $c^\star \leftarrow (\!|c^\star|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk})$;
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_{\mathcal{A}}}(c^\star, \sigma)$;
  $b \xleftarrow{\$} \{0,1\}$
```

In this experiment, the value of b is uniformly distributed and obviously independent of $\bar{b}$, and thus

$$\Pr\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : b = \bar{b}\right] = 1/2$$

Case [$\mathsf{Rnd}(\ell, \vec{H})$]**:** Since $r \notin \mathcal{R}(c^\star)$, we can defer initializing r to the end of the CPA experiment (call G the resulting game). We have

$$\Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)\right]\!\right] = \Pr\left[\!\left[G : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)\right]\!\right]$$

Let C be the context witnessing the deducibility of r from $\vec{e} \,\|\, \mathcal{R}(c^\star) \,\|\, m$. Since the value of r is chosen uniformly and independently from the queries that the adversary makes to oracles, $\mathcal{R}(c^\star)$, and m_b, we have that for any $\vec{e} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}$,

$$\Pr\left[G : (\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{e} \,\|\, \mathcal{R}(c^\star) \,\|\, m_b) = r\right] = 2^{-|r|}$$

Moreover, from the union bound and because $|\boldsymbol{L}_{H_i}| \le q_{H_i}, i = 1, \ldots, n$ is a post-condition of G,

$$\begin{aligned}
\Pr\left[\!\left[G : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)\right]\!\right] &\le \Pr\left[G : \exists \vec{e} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}.\ (\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{e} \,\|\, \mathcal{R}(c^\star) \,\|\, m_b) = r\right] \\
&\le \sum_{\vec{e} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}} \Pr\left[G : (\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, \vec{e} \,\|\, \mathcal{R}(c^\star) \,\|\, m_b) = r\right] \\
&\le \prod_{i=1}^{n} q_{H_i} \times 2^{-|r|}
\end{aligned}$$

Case $[\mathsf{OW}_k^\ell(\mathsf{f},\vec{H})]$**:** Assume $\vec{e} \,\|\, r_2 \,\|\, m \vdash [r_1]_k^\ell$ and $\mathsf{f}(r_1) \,\|\, r_2 \,\|\, m \vdash c^\star$ and call C_1 and C_2 the contexts witnessing these deducibilities. It suffices to show that for any adversary $\mathcal{A}$ against CPA, there exists an inverter $\mathcal{I}$ executing within time $t_\mathcal{A} + T(C_2) + q_{\vec{H}} \times T(C_1)$ such that

$$\Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)\right]\!\right] \leq \mathbf{Succ}_{\Theta_\mathsf{f}}^{\mathsf{s\text{-}pd\text{-}OW}}(k, \ell, q_{\vec{H}}, \mathcal{I})$$

where $q_{\vec{H}} = \prod_{i=1}^n q_{H_i}$. Given an adversary $\mathcal{A}$, we construct an inverter $\mathcal{I}$ that achieves this bound:

```
Procedure $\mathcal{I}(y, pk) \overset{\text{def}}{=}$
  foreach $\mathsf{g} \in \mathcal{F} \setminus \{\mathsf{f}\}$ do $(pk_g, sk_g) \leftarrow \mathcal{KG}_g$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\boldsymbol{H} \leftarrow \emptyset$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r_2 \overset{\$}{\leftarrow} \{0,1\}^{|r_2|}$;
  $(\!|c^\star \lhd C_2|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, y \,\|\, r_2 \,\|\, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
  $S \leftarrow \emptyset$;
  foreach $\vec{e} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}$ do
    $(\!|x \lhd C_1|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{e} \,\|\, r_2 \,\|\, m_b)$; $S \leftarrow S \cup \{x\}$
  return $S$
```

```
Procedure $H_\mathcal{A}(x) \overset{\text{def}}{=}$
  if $|\boldsymbol{L}_H| < q_H$ then $\boldsymbol{L}_H \leftarrow x :: \boldsymbol{L}_H$; return $H(x)$
  else return $\perp$

Procedure $H(x) \overset{\text{def}}{=}$
  if $x \notin \mathsf{dom}(\boldsymbol{H})$ then $\boldsymbol{H}[x] \overset{\$}{\leftarrow} \{0,1\}^\ell$;
  return $\boldsymbol{H}[x]$
```

The inverter simulates an environment for $\mathcal{A}$ that is indistinguishable from the real CPA experiment: it simulates hash oracles exactly as in the CPA experiment, storing the queries the adversary makes. The inverter embeds its own challenge $f_{pk}(s)$ into the challenge ciphertext $c^\star$ given to $\mathcal{A}$, in such a way that if $\mathcal{A}$ asks queries $H_1(e_1), \ldots, H_n(e_n)$, then from $\vec{e}$, $\mathcal{I}$ can recover a partial pre-image of $f_{pk}(s)$. The running time of the inverter is indeed $t_\mathcal{A} + T(C_2) + q_{\vec{H}} \times T(C_1)$.
Recall that the OW_k^ℓ experiment is

$$(pk, sk) \leftarrow \mathcal{KG};\ s \overset{\$}{\leftarrow} \{0,1\}^{|\mathsf{f}|};\ S \leftarrow \mathcal{I}(f_{pk}(s), pk)$$

Inlining $\mathcal{I}$ in this experiment, one obtains an experiment that is essentially equivalent to $\mathsf{CPA}_\mathcal{H}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A})$:

```
Game $G \overset{\text{def}}{=}$
  $(pk, sk) \leftarrow \mathcal{KG}_f$;
  $s \overset{\$}{\leftarrow} \{0,1\}^{|\mathsf{f}|}$;
  foreach $\mathsf{g} \in \mathcal{F} \setminus \{\mathsf{f}\}$ do $(pk_g, sk_g) \leftarrow \mathcal{KG}_g$;
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\boldsymbol{H} \leftarrow \emptyset$;
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
  $b \overset{\$}{\leftarrow} \{0,1\}$;
  $r_2 \overset{\$}{\leftarrow} \{0,1\}^{|r_2|}$;
  $(\!|c^\star \lhd C_2|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, f_{pk}(s) \,\|\, r_2 \,\|\, m_b)$;
  $\bar{b} \leftarrow \mathcal{A}_2(c^\star, \sigma)$;
  $S \leftarrow \emptyset$;
  foreach $\vec{e} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}$ do
    $(\!|x \lhd C_1|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{e} \,\|\, r_2 \,\|\, m_b)$; $S \leftarrow S \cup \{x\}$
```

Therefore, from Lemmas 16 and 21, and since $|S| \leq q_{\vec{H}}$ is a post-condition of OW_k^ℓ,

$$\begin{aligned}
&\Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)\right]\!\right] \\
&\quad = \Pr\left[\!\left[\mathsf{CPA}_{\mathcal{H}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Ask}(H_1, e_1) \wedge \cdots \wedge \mathsf{Ask}(H_n, e_n)\right]\!\right] \\
&\quad \leq \Pr\left[\begin{array}{ll} \mathsf{CPA}_{\mathcal{H}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}); S \leftarrow \emptyset; & \\ \mathsf{foreach}\ \vec{e} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}\ \mathsf{do} & : [r_1]_k^\ell \in S \\ \quad (\!|x \lhd C_1|\!)^{\vec{H}}(\vec{pk}, \vec{sk}, \vec{e} \,\|\, r_2 \,\|\, m_b);\ S \leftarrow S \cup \{x\} & \end{array}\right] \\
&\quad = \Pr\left[\mathsf{OW}_k^\ell : [r_1]_k^\ell \in S\right] \\
&\quad = \mathbf{Succ}_{\Theta_{\mathsf{f}}}^{\mathsf{s\text{-}pd\text{-}OW}}(k, \ell, q_{\vec{H}}, \mathcal{I})
\end{aligned}$$

Theorem 23 (Soundness of CPA logic)**.** *If ∇ is a derivation of $\vDash_{1/2} c^\star : \mathsf{Guess}$ under Γ, then*

$$\mathbf{Adv}_{\Pi}^{\mathsf{CPA}}(t_{\mathcal{A}}, \vec{q}) \leq 2\, \mathcal{B}_{(t_A, \vec{q})}(\nabla) - 1$$

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

Proof. From Lemma 22, for some CPA adversary $\mathcal{A}$,

$$\begin{aligned}
\mathbf{Adv}_{\Pi}^{\mathsf{CPA}}(t_{\mathcal{A}}, \vec{q}) &= 2 \Pr\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : b = \bar{b}\right] - 1 \\
&= 2 \Pr\left[\!\left[\mathsf{CPA}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \mathsf{Guess}\right]\!\right] - 1 \\
&\leq 2\, \mathcal{B}_{(t_A, \vec{q})}(\nabla) - 1
\end{aligned}$$

The derivation ∇ must contain exactly one application of rule [Ind]. This is so because no branching rule contains in the premise more than one judgment with $1/2$ as probability tag. Moreover, for a derivation ∇' that does not contain an application of [Ind], the bound $\mathcal{B}_{(t_A, \vec{q})}(\nabla')$ is negligible in the security parameter if $t_{\mathcal{A}}$ and all q_H in $\vec{q}$ are polynomial in the security parameter. This can be proved by induction on the structure of ∇':

Cases [Opt]**,** [Perm]**,** [Split]**,** [Merge]**,** [Sub]**, and** [$\mathsf{Fail}_{1,2}$] Follow from the inductive hypothesis.

Case [$\mathsf{Rnd}(\ell, H)$] Since ℓ is interpreted as a non-constant polynomial, and all q_H in $\vec{q}$ are polynomial in the security parameter, the bound $q_{\vec{H}} \times 2^{-\ell}$ is negligible.

Case [$\mathsf{OW}_k^\ell(\mathsf{f}, \vec{H})$] Observe first that since $t_{\mathcal{A}}$ and all q_H in $\vec{q}$ are polynomial in the security parameter, $t_{\mathcal{I}}$ is also polynomial. Hence, since $(\mathsf{f}, k, \ell) \in \Gamma$, the bound $\mathbf{Succ}_{\Theta_{\mathsf{f}}}^{\mathsf{s\text{-}pd\text{-}OW}}(k, \ell, q_{\vec{H}}, t_{\mathcal{I}})$ is negligible in the security parameter.

Therefore, $\mathcal{B}_{(t_A, \vec{q})}(\nabla)$ is of the form $1/2 + \nu$ for some negligible function ν, and $2\, \mathcal{B}_{(t_A, \vec{q})}(\nabla) - 1 = \nu$ is negligible. □

A.4 Soundness of CCA logic

All definitions and proofs in this section are relative to fixed finite sets $\mathcal{F}$ and $\mathcal{H}$ of (typed) trapdoor permutation and hash function symbols. We assume that all expressions and events under consideration only contain trapdoor permutation and hash function symbols in these sets.

We first define the CCA experiment corresponding to a given CCA judgment.

Definition 24 (CCA experiment)**.** *Let e be a well-typed algebraic expression such that $\mathcal{X}(e) \subseteq \{m\}$. Let $D = (\mathsf{find}\ x_1\ \mathsf{in}\ \boldsymbol{L}_{H_1}, \ldots, \mathsf{find}\ x_k\ \mathsf{in}\ \boldsymbol{L}_{H_k}, \diamond : T \rhd t)$ be a well-typed decryption oracle such that $\mathcal{X}(D) \subseteq \{c\}$*

and $\mathcal{R}(D) = \emptyset$. Let $\mathcal{R}$ be a set of random bitstring symbols such that $\mathcal{R}(e) \subseteq \mathcal{R}$. Given a CCA *adversary $\mathcal{A}$ and $\vec{q} \in \mathbb{N}^{|\mathcal{H}|+1}$, define the program* $\mathsf{CCA}(c^\star, D, \mathcal{R}, \vec{q}, \mathcal{A})$ *as follows:*

```
Game $\mathsf{CCA}(c^\star, D, \mathcal{R}, \vec{q}, \mathcal{A}) \overset{\text{def}}{=}$
  $\boldsymbol{L}_\mathcal{D} \leftarrow \mathsf{nil};\ n_\mathcal{D} \leftarrow 0;$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f;$
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil};\ \mathsf{init}_H;$
  foreach $r \in \mathcal{R}$ do $r \overset{\$}{\leftarrow} \{0,1\}^{|r|};$
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}, \mathcal{D}_1}(\vec{pk});$
  $b \overset{\$}{\leftarrow} \{0,1\};$
  $c^\star \leftarrow (\!|e|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b);$
  $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_\mathcal{A}, \mathcal{D}_2}(c^\star, \sigma)$

Procedure $H_\mathcal{A}(x) \overset{\text{def}}{=}$
  if $|\boldsymbol{L}_H| < q_H$ then $\boldsymbol{L}_H \leftarrow x :: \boldsymbol{L}_H;$ return $H(x)$
  else return $\perp$

Procedure $H(x) \overset{\text{def}}{=}$ return $\boldsymbol{H}[x]$

Procedure $\mathcal{D}(c) \overset{\text{def}}{=}$
  foreach $x_1 \in \boldsymbol{L}_{H_1}, \ldots, x_k \in \boldsymbol{L}_{H_k}$ do
    if $(\!|T|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x}, \boldsymbol{L}_{\vec{H}})$ then
      return $(\!|t|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x})$
  return $\perp$

Procedure $\mathcal{D}_1(c) \overset{\text{def}}{=}$
  if $n_\mathcal{D} < q_\mathcal{D}$ then $n_\mathcal{D} \leftarrow n_\mathcal{D} + 1;$ return $\mathcal{D}(c)$
  else return $\perp$

Procedure $\mathcal{D}_2(c) \overset{\text{def}}{=}$
  if $n_\mathcal{D} < q_\mathcal{D} \wedge c \neq c^\star$ then
    $n_\mathcal{D} \leftarrow n_\mathcal{D} + 1;\ \boldsymbol{L}_\mathcal{D} \leftarrow c :: \boldsymbol{L}_\mathcal{D};$ return $\mathcal{D}(c);$
  else return $\perp$
```

The adversary $\mathcal{A}$ is represented as a pair of procedures $(\mathcal{A}_1, \mathcal{A}_2)$ that are given access to a hash oracle H through a wrapper $H_\mathcal{A}$ that stores queries in a list $\boldsymbol{L}_H$ and limits the number of distinct queries to q_H. Additionally $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) is given access to a decryption oracle $\mathcal{D}_1$ (resp. $\mathcal{D}_2$) that limits the number of queries to $q_\mathcal{D}$; in addition $\mathcal{D}_2$ rejects $c^\star$ as a query and maintains a list $\boldsymbol{L}_\mathcal{D}$ of queries made by $\mathcal{A}_2$.

The soundness proof also requires an extended version $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}$ of the CPA experiment where the adversary returns a list $\boldsymbol{L}_\mathcal{D}$ of decryption queries in addition to its guess $\bar{b}$ for b. This extension is required for the proof of rule [Pub] where the probability of an event in the CCA experiment with an adversary $\mathcal{A}$ is related to the probability of the same event in the $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}$ experiment with an adversary $\mathcal{B}$ that simulates a decryption oracle for $\mathcal{A}$. Since CCA events might refer to $\boldsymbol{L}_\mathcal{D}$, the simulator $\mathcal{B}$ has to forward $\mathcal{A}$'s guess and return $\boldsymbol{L}_\mathcal{D}$.

The game $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}(e, \mathcal{R}, \vec{q}, \mathcal{A})$ is defined as follows:

```
Game $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}(e, \mathcal{R}, \vec{q}, \mathcal{A}) \overset{\text{def}}{=}$
  foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f;$
  foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil};\ \mathsf{init}_H;$
  foreach $r \in \mathcal{R}$ do $r \overset{\$}{\leftarrow} \{0,1\}^{|r|};$
  $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_\mathcal{A}}(\vec{pk});$
  $b \overset{\$}{\leftarrow} \{0,1\};$
  $c^\star \leftarrow (\!|e|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b);$
  $(\boldsymbol{L}_\mathcal{D}, \bar{b}) \leftarrow \mathcal{A}_2^{\vec{H}_\mathcal{A}}(c^\star, \sigma)$
```

We also define a game $\mathsf{CPA}_{\mathsf{L},\mathcal{G}}^{\boldsymbol{L}_\mathcal{D}}(e, \mathcal{R}, \vec{q}, \mathcal{A})$ analogously to $\mathsf{CPA}_\mathcal{G}$. Before proving the soundness of the CCA rules, we prove the soundness of the extended set of CPA rules with respect to an interpretation defined in terms of $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}$. To achieve this, we extend the semantics of events to handle events of the form T and $\exists x \in \boldsymbol{L}_\mathcal{D}.\ T$ in the expected way.

Lemma 25 (Soundness of extended CPA logic)**.** *Let $c^\star$ be a well-typed algebraic expression with $\mathcal{X}(c^\star) \subseteq \{m\}$, ϕ an event of the* CCA *logic, and $\hat{p}$ a probability tag. Let ∇ be a derivation of $\vDash_{\hat{p}} c^\star : \phi$ from the rules of Figure 3 and the* CPA *rules of Figure 4. For any set of random bitstrings $\mathcal{R}$ such that $\mathcal{R}(e, \phi) \subseteq \mathcal{R}$, any $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}$ adversary $\mathcal{A}$, and any $\vec{q} \in \mathbb{N}^{|\mathcal{H}|+1}$,*

$$\Pr \left[\!\left[\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \leq \mathcal{B}_{(t_A, \vec{q})}(\nabla)$$

Proof. By structural induction on the derivation ∇.

Case $[\mathsf{Opt}], [\mathsf{Perm}], [\mathsf{Merge}], [\mathsf{Split}], [\mathsf{Sub}], [\mathsf{Fail}_{1,2}], [\mathsf{Ind}]$**:** The proofs of Lemma 22 can be naturally extended to events of the form T and $\exists x \in \boldsymbol{L}_\mathcal{D}.\ T$, interpreted in the experiment $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A})$.

Case $[\mathsf{Eqs}]$**:** Since $r \notin \mathcal{R}(c^\star, e)$, we can defer sampling it till the en of game $\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A})$. Call the resulting game G. We have that

$$\Pr \left[\!\left[\mathsf{CPA}^{\boldsymbol{L}_\mathcal{D}}(c^\star, \mathcal{R}, \vec{q}, \mathcal{A}) : e = r\right]\!\right] = \Pr \left[\!\left[G : e = r\right]\!\right].$$

Since r is uniformly distributed and obviously independent from all variables e may depend on, we conclude that

$$\Pr \left[\!\left[G : e = r\right]\!\right] \leq 2^{-|r|}.$$

Case (extended) $[\mathsf{Rnd}]$ **and** $[\mathsf{OW}]$**:** The proof follows the same argument as in the original rules (the bounds are the same). Note that the contexts witnessing the deducibility relations in the premises of these rules cannot refer directly to ciphertexts in $\boldsymbol{L}_\mathcal{D}$, and thus do not require c as an argument. $\square$

The soundness of the CCA logic relies on the soundness of judgments of the form $\vDash \mathsf{negl}_r(T)$, which we prove next.

Lemma 26 (Soundness of negl logic)**.** *Let r be a random bitstring symbol and T a well-typed test with $\mathcal{X}(T) \subseteq \{c\}$ and $\mathcal{R}(T) \subseteq \{r\}$. Let ∇ be a derivation of $\vDash \mathsf{negl}_r(T)$. For all t_A, $\vec{q} \in \mathbb{N}^{|\mathcal{H}|+1}$ and $\boldsymbol{L}_{\vec{H}}$ such that $|\boldsymbol{L}_H| \leq q_H$,*

$$\Pr \left[r \leftarrow \{0,1\}^{|r|} : (\!|T|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \leq \mathcal{B}_{(t_A, \vec{q})}(\nabla)$$

Proof. By structural induction on the derivation ∇.

Case $[\mathsf{PAnd}]$**:** Assume wlog $i = 1$. It suffices to prove

$$\Pr \left[r \leftarrow \{0,1\}^{|r|} : (\!|T_1 \wedge T_2|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \leq \Pr \left[r \leftarrow \{0,1\}^{|r|} : (\!|T_1|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right]$$

which follows from the fact that

$$(\!|T_1 \wedge T_2|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}}) = (\!|T_1|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}}) \wedge (\!|T_2|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})$$

Case $[\mathsf{PEqs}]$**:** Let C be a context witnessing the deducibility $c \,\|\, e \vdash^\star [r]_k^\ell$. Knowing that $r \notin \mathcal{R}(e')$, we have to show that

$$\Pr \left[r \leftarrow \{0,1\}^{|r|} : (\!|e = e'|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \leq 2^{-\ell}$$

Note that if $(\!|e = e'|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}}) = \mathsf{true}$, then from Lemma 16

$$(\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, e') = (\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, e) = [r]_k^\ell$$

Since $r \notin \mathcal{R}(c \,\|\, e')$ (c is a variable) and $\mathcal{R}(C) = \emptyset$, $(\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, e')$ is independent of r, and thus

$$\Pr \left[r \leftarrow \{0,1\}^{|r|} : (\!|e = e'|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \leq \Pr \left[r \leftarrow \{0,1\}^{|r|} : (\!|C|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, e') = [r]_k^\ell\right] = 2^{-\ell}$$

Case [PRnd]**:** Let C be a context witnessing the deducibility $c \,\|\, e \vdash^\star [r]_k^\ell$. We have to show that

$$\Pr\left[r \leftarrow \{0,1\}^{|r|} : \llparenthesis \mathsf{Ask}(H,e) \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \leq q_H \times 2^{-\ell}$$

From Lemma 16, we have $\llparenthesis C \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, e) = [r]_k^\ell$ Since $c \,\|\, \boldsymbol{L}_H[i]$ is independent of r for $0 < i \leq q_H$, we have

$$\begin{aligned}
&\Pr\left[r \leftarrow \{0,1\}^{|r|} : \llparenthesis \mathsf{Ask}(H,e) \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \\
&\quad \leq \Pr\left[r \leftarrow \{0,1\}^{|r|} : \llparenthesis C \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, \boldsymbol{L}_H[1]) = [r]_k^\ell \vee \ldots \vee \llparenthesis C \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, \boldsymbol{L}_H[q_H]) = [r]_k^\ell\right] \\
&\quad \leq \sum_{i=1}^{q_H} \Pr\left[r \leftarrow \{0,1\}^{|r|} : \llparenthesis C \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c \,\|\, \boldsymbol{L}_H[i]) = [r]_k^\ell\right] = q_H \times 2^{-\ell} \qquad \square
\end{aligned}$$

We now have all the tools to prove the soundness of the CCA logic.

Lemma 27. *Let $c^\star$ be a well-typed algebraic expression with $\mathcal{X}(c^\star) \subseteq \{m\}$ and D a well-defined decryption oracle with $D = (\mathsf{find}\ x_1\ \mathsf{in}\ \boldsymbol{L}_{H_1}, \ldots, \mathsf{find}\ x_k\ \mathsf{in}\ \boldsymbol{L}_{H_k}, \diamond : T \rhd t)$ such that $\mathcal{X}(D) \subseteq \{c\}$ and $\mathcal{R}(D) = \emptyset$. Let ϕ be an event of the CCA logic and $\hat{p}$ a probability tag. Let ∇ be a derivation of $\vDash_{\hat{p}} (c^\star, D) : \phi$ from the rules of Figure 4. Then, for any CCA adversary $\mathcal{A}$ and $\vec{q} \in \mathbb{N}^{|\mathcal{H}|+1}$,*

$$\Pr\left[\!\left[\mathsf{CCA}(c^\star, D, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \leq \mathcal{B}_{(t_A, \vec{q})}(\nabla)$$

Proof. By structural induction on the derivation ∇.

Case [False]**:** Since $c^\star \notin \boldsymbol{L}_{\mathcal{D}}$ is enforced by the wrapper $\mathcal{D}_2$,

$$\Pr\left[\mathsf{CCA}(c^\star, D, \mathcal{R}, \vec{q}, \mathcal{A}) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\, c = c^\star\right] = \Pr\left[\mathsf{CCA}(c^\star, D, \mathcal{R}, \vec{q}, \mathcal{A}) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\, c = c^\star \wedge c \neq c^\star\right] = 0\,.$$

Case [Find]**:** It suffices to show that

$$\begin{aligned}
&\Pr\left[\!\left[\mathsf{CCA}(c^\star, F : T\,\{e/x\} \wedge \mathsf{Ask}(H,e) \rhd t\,\{e/x\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \leq \\
&\Pr\left[\!\left[\mathsf{CCA}(c^\star, \mathsf{find}\ x\,\mathsf{in}\ \boldsymbol{L}_H, F : T \wedge x = e \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right]
\end{aligned}$$

The two experiments above are identical except for the implementation of the decryption oracle. We show that the decryption oracles in both experiments behave identically. Suppose F is of the form

$$\mathsf{find}\ x_1\ \mathsf{in}\ \boldsymbol{L}_{H_1}, \ldots, \mathsf{find}\ x_k\ \mathsf{in}\ \boldsymbol{L}_{H_k}, \diamond$$

and let $\vec{x} = (x_1, \ldots, x_k)$. The corresponding decryption oracles are, respectively:

```
Procedure $\mathcal{D}_L(c) \overset{\text{def}}{=}$
  foreach $x_1 \in \boldsymbol{L}_{H_1}, \ldots, x_k \in \boldsymbol{L}_{H_k}$ do
    if $\llparenthesis T\,\{e/x\} \wedge \mathsf{Ask}(H,e) \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x}, \boldsymbol{L}_{\vec{H}})$ then
      return $\llparenthesis t\,\{e/x\} \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x})$;
  return $\perp$
```

```
Procedure $\mathcal{D}_R(c) \overset{\text{def}}{=}$
  foreach $x \in \boldsymbol{L}_H, x_1 \in \boldsymbol{L}_{H_1}, \ldots, x_k \in \boldsymbol{L}_{H_k}$ do
    if $\llparenthesis T \wedge x = e \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, x, \vec{x}, \boldsymbol{L}_{\vec{H}})$ then
      return $\llparenthesis t \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, x, \vec{x})$;
  return $\perp$
```

We assume that we run these two programs in the same state and with the same argument. We want to show that both yield the same result.

From Lemma 13, the foreach loop in $\mathcal{D}_L$ is equivalent to

```
foreach $x_1 \in \boldsymbol{L}_{H_1}, \ldots, x_k \in \boldsymbol{L}_{H_k}$ do
  let $x = \langle\!\langle e \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x})$ in
    if $\langle\!\langle T \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, x, \vec{x}, \boldsymbol{L}_{\vec{H}}) \wedge x \in \boldsymbol{L}_H$ then
      return $\langle\!\langle t \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, x, \vec{x})$;
return $\perp$
```

while the loop in $\mathcal{D}_R$ is equivalent to

```
foreach $x \in \boldsymbol{L}_H, x_1 \in \boldsymbol{L}_{H_1}, \ldots, x_k \in \boldsymbol{L}_{H_k}$ do
  let $x' = \langle\!\langle e \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x})$ in
    if $x = x' \wedge \langle\!\langle T \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, x, \vec{x}, \boldsymbol{L}_{\vec{H}})$ then
      return $\langle\!\langle t \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, x, \vec{x})$;
return $\perp$
```

It is easy to see that both are equivalent. Both tests succeed iff there exists $\vec{x} \in \boldsymbol{L}_{H_1} \times \cdots \times \boldsymbol{L}_{H_n}$ such that $\langle\!\langle e \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x}) \in \boldsymbol{L}_H$ and $\langle\!\langle T \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \langle\!\langle e \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \vec{x}), \vec{x}, \boldsymbol{L}_{\vec{H}})$.
Hence, we conclude

$$\begin{aligned}&\Pr \left[\!\left[\mathsf{CCA}(c^\star, F : T\{e/x\} \wedge \mathsf{Ask}(H, e) \rhd t\{e/x\}, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] = \\ &\Pr \left[\!\left[\mathsf{CCA}(c^\star, \mathsf{find}\ x\ \mathsf{in}\ \boldsymbol{L}_H, F : T \wedge x = e \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right]\end{aligned}$$

Case [Pub]**:** Assume we have a derivation ∇ of $\vDash_{\hat{p}} c^\star : \phi$ in the extended CPA logic. From lemma 25, for any $\mathsf{CPA}^{\boldsymbol{L}_{\mathcal{D}}}$ adversary $\mathcal{S}$ and $\vec{q}^{\mathcal{S}} \in \mathbb{N}^{|\mathcal{H}|}$,

$$\Pr \left[\!\left[\mathsf{CPA}^{\boldsymbol{L}_{\mathcal{D}}}(c^\star, \mathcal{R}, \vec{q}^{\mathcal{S}}, \mathcal{S}) : \phi\right]\!\right] \leq \mathcal{B}_{(t_{\mathcal{S}}, \vec{q}^{\mathcal{S}})}(\nabla).$$

Knowing that $F : T \rhd u$ is public, i.e. it does not use f^{-1}, it suffices to show that for any CCA adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ that executes in time

$$t_{\mathcal{S}} = t_{\mathcal{A}} + q_{\mathcal{D}} \times (\mathsf{T}(u) + \mathsf{T}(T) \times \prod_{H_j\ \mathrm{in}\ F} q_{H_j})$$

and such that

$$\Pr \left[\!\left[\mathsf{CCA}(c^\star, F : T \rhd u, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \leq \Pr \left[\!\left[\mathsf{CPA}^{\boldsymbol{L}_{\mathcal{D}}}(c^\star, \mathcal{R}, \vec{q}^{\mathcal{S}}, \mathcal{S}) : \phi\right]\!\right]$$

where

$$q^{\mathcal{S}}_{H_i} = q_{H_i} + q_{\mathcal{D}} \times (\mathsf{Q}_i(u) + \mathsf{Q}_i(T) \times \prod_{H_j\ \mathrm{in}\ F} q_{H_j}).$$

Recall that $\mathsf{T}(\cdot)$ and $\mathsf{Q}_i(\cdot)$ give, respectively, an upper bound on the time and number of queries to H_i needed to evaluate a test or an expression. The simulator $\mathcal{S}$ we exhibit just forwards its inputs to $\mathcal{A}$ and outputs $\mathcal{A}$'s responses. It answers $\mathcal{A}$'s hash queries by querying its own oracles, and simulates $\mathcal{D}_1$ and $\mathcal{D}_2$ using the same wrappers $\mathcal{D}_1$ and $\mathcal{D}_2$ and procedure $\mathcal{D}$ as in the CCA experiment (note that $\vec{sk}$ need not be used, since the decryption oracle $F : T \rhd u$ is public):

```
Procedure $\mathcal{D}(c) \stackrel{\text{def}}{=}$
  foreach $x_1 \in \boldsymbol{L}_{H_1}, \ldots, x_k \in \boldsymbol{L}_{H_k}$ do
    if $\langle\!\langle T \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, c, \vec{x}, \boldsymbol{L}_{\vec{H}})$ then return $\langle\!\langle u \rangle\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, c, \vec{x})$
  return $\perp$
```

Since the simulator needs to answer at most number $q_{\mathcal{D}}$ decryption queries, it suffices to show that we can simulate each query in time

$$\mathsf{T}(u) + \mathsf{T}(T) \times \prod_{H_j \text{ in } F} q_{H_j}$$

and making at most

$$\mathsf{Q}_i(u) + \mathsf{Q}_i(T) \times \prod_{H_j \text{ in } F} q_{H_j}$$

queries to H_i. For each query, the above procedure evaluates the test T at most $q_{\vec{H}}$ times, for a total execution time bounded by $q_{\vec{H}}\,\mathsf{T}(T)$, and makes at most $q_{\vec{H}}\,\mathsf{Q}_i(T)$ extra queries to H_i. It also evaluates u at most once, in time bounded by $\mathsf{T}(u)$ and making at most $\mathsf{Q}_i(u)$ more queries to H_i.
To conclude, observe that inlining the definition of $\mathcal{S}$ in game $\mathsf{CPA}^{\boldsymbol{L}_{\mathcal{D}}}(c^\star, \mathcal{R}, \vec{q}^{\mathcal{S}}, \mathcal{S})$ results in a game that is equivalent to the original CCA experiment. This is so, because $\mathcal{S}$ is allotted as many queries as it can ever need to simulate $\mathcal{D}$ for $\mathcal{A}$.

Case [Bad]**:** Recall the rule is defined as

$$\frac{\begin{array}{c}\mathcal{ST}_H(c^\star) = \{H(e^\star)\} \quad \mathcal{ST}_H(T,t) = \{H(e)\} \quad \vDash \mathsf{negl}_r(T\,\{r/H(e)\}) \\ \vDash_{\hat{p}} (c^\star, T \wedge \mathsf{Ask}(H,e) \rhd t) : \phi \quad \vDash_0 (c^\star, T \wedge \mathsf{Ask}(H,e) \rhd t) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ T \wedge e^\star = e\end{array}}{\vDash_{\hat{p}} (c^\star, T \rhd t) : \phi}\,[\mathsf{Bad}]$$

and when $L_\nabla = [\mathsf{Bad}]$, $\mathcal{B}(\nabla) = \mathcal{B}(\nabla_1) + \mathcal{B}(\nabla_2) + q_{\mathcal{D}}\,\mathcal{B}(\nabla_3)$. It suffices to show

$$\begin{aligned}\Pr\left[\!\left[\mathsf{CCA}(c^\star, T \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \le{} & \Pr\left[\!\left[\mathsf{CCA}(c^\star, T \wedge \mathsf{Ask}(H,e) \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \\ & + \Pr\left[\!\left[\mathsf{CCA}(c^\star, T \wedge \mathsf{Ask}(H,e) \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ T \wedge e = e^\star\right]\!\right] \\ & + q_{\mathcal{D}} \Pr\left[r \leftarrow \{0,1\}^{|r|} : \langle\!| T |\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right].\end{aligned}$$

We first present an outline of the proof, and then give detailed justifications for each inequality. The sequence of games and events used throughout the proof is shown in Figures 10 and 11.
We start with $G_0 \stackrel{\text{def}}{=} \mathsf{CCA}(c^\star, T \rhd t, \mathcal{R}, \vec{q}, \mathcal{A})$ and obtain G_1 from G_0 by instrumenting the decryption oracle $\mathcal{D}$ to maintain history variables storing for each query, the queried ciphertext, the current value of $\boldsymbol{L}_{\vec{H}}$, and the current value of m_b. We also sample an integer between 0 and $q_{\mathcal{D}} - 1$ which will be used later on. We have

$$\Pr\left[\!\left[G_0 : \phi\right]\!\right] \le \Pr\left[\!\left[G_1 : \phi\right]\!\right]. \tag{1}$$

We obtain G_2 from G_1 by adding the conjunct $\mathsf{Ask}(H,e) \vee e = e^\star$ to the test T. Note that $e = e^\star$ is always false in the first stage of the experiment since $e^\star$ is undefined. It follows that

$$\Pr\left[\!\left[G_1 : \phi\right]\!\right] \le \Pr\left[\!\left[G_2 : \phi\right]\!\right] + \Pr\left[G_2 : \psi\right] \tag{2}$$

where ψ holds if there is a decryption query c such that T succeeds but neither $\mathsf{Ask}(H,e)$ nor $e = e^\star$ holds, i.e. the bitstring e has not been queried to the random oracle before the decryption query. We bound each probability separately.

- To bound $\Pr\left[\!\left[G_2 : \phi\right]\!\right]$, we obtain G_3' from G_2 by replacing $\mathsf{Ask}(H,e) \vee e = e^\star$ in the test with $\mathsf{Ask}(H,e)$. It holds that

$$\Pr\left[\!\left[G_2 : \phi\right]\!\right] \le \Pr\left[\!\left[G_3' : \phi\right]\!\right] + \Pr\left[G_3' : \xi\right] \tag{3}$$

where ξ holds if there is a query such that $T \wedge e = e^\star$ holds. Furthermore,

$$\Pr\left[\!\left[G_3' : \xi\right]\!\right] \le \Pr\left[G_3' : \langle\!|\exists c \in \boldsymbol{L}_{\mathcal{D}}.\ T \wedge e = e^\star|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}}, m_b)\right] \tag{4}$$

since the test $e = e^\star$ always fails during the first phase, for all i such that the body of ξ is true, there exists j with $\boldsymbol{L}_{\mathcal{D}}[j] = hist_c[i]$. Additionally, note that if the test succeeds for $hist_{\vec{\boldsymbol{L}_H}}[i]$, then it also succeeds for $\boldsymbol{L}_{\vec{H}}$ since for all i and for all $H \in \mathcal{H}$, it holds that $hist_{\boldsymbol{L}_H}[i] \subseteq \boldsymbol{L}_H$. We conclude this case by observing that

$$\Pr\left[\!\left[G_3' : \phi\right]\!\right] = \Pr\left[\!\left[\mathsf{CCA}(c^\star, T \wedge \mathsf{Ask}(H, e) \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \phi\right]\!\right] \tag{5}$$

and

$$\begin{aligned}&\Pr\left[G_3' : \langle\!|\exists c \in \boldsymbol{L}_{\mathcal{D}}.\ T \wedge e = e^\star|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}}, m_b)\right] \\ &= \Pr\left[\!\left[\mathsf{CCA}(c^\star, T \wedge \mathsf{Ask}(H, e) \rhd t, \mathcal{R}, \vec{q}, \mathcal{A}) : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ T \wedge e = e^\star\right]\!\right].\end{aligned} \tag{6}$$

- To bound $\Pr\left[\!\left[G_2 : \psi\right]\!\right]$, we first use i as a guess of the query for which ψ is true. Then, we obtain

$$\Pr\left[G_2 : \psi\right] \le q_{\mathcal{D}} \times \Pr\left[G_2 : \psi_i\right] \tag{7}$$

where ψ_i is true if the body of ψ holds for the i-th query. In the next step, we obtain G_3 from G_2 by modifying $\mathcal{D}$ to behave exacly as in G_2, except for the i-th query where $\mathcal{D}$ updates the history variables, stores the argument c as ci, and halts the game. It holds that

$$\Pr\left[G_2 : \psi_i\right] = \Pr\left[G_3 : \psi_i\right] = \Pr\left[G_3 : \psi'\right] \tag{8}$$

where ψ' is identical to ψ_i except that it refers to ci, m_b, and $\boldsymbol{L}_{\vec{H}}$ directly instead of using the history variables. Additionally, since m_b is undefined if the i-th query is made in the first phase, this check is now implicitly included in the comparison $e = e^\star$. We then obtain G_4 from G_3 by hoisting $H(\langle\!|e^\star|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b))$ out of the challenge ciphertext computation, hoisting $H(\langle\!|e|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c))$ out of T and t, and hoisting $H(\langle\!|e|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci))$ out of ψ'. We can now switch to a lazy random oracle implementation for H to obtain G_5. It holds that

$$\Pr\left[G_3 : \psi'\right] = \Pr\left[G_4 : \psi'\right] = \Pr\left[G_4 : \rho\right] = \Pr\left[G_5 : \rho\right] \tag{9}$$

where ρ is the result of hoisting $H(\langle\!|e|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci)$ out of ψ'. Note that the resulting decryption function $\mathcal{D}$ does not perform any *new* queries to H since the test $\mathsf{Ask}(H, e) \vee e = e^\star$ ensures that e has already been queried by the adversary or in the computation of $c^\star$. We can therefore obtain G_6 by replacing the procedure call $H(\langle\!|e|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci))$ with a random sampling. We can now prove that for all $\boldsymbol{L}_{\vec{H}}$ and for all $\boldsymbol{L}_{\mathcal{D}}$ whose lengths are bounded by $\vec{q}$,

$$\begin{aligned}\Pr\left[G_5 : \rho\right] &= \Pr\left[G_6 : \rho\right] \\ &\le \Pr\left[r \leftarrow \{0,1\}^{|r|} : \exists c \in \boldsymbol{L}_{\mathcal{D}}.\ \langle\!|T\{r/H(e)\}|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right] \\ &\le q_{\mathcal{D}} \Pr\left[r \leftarrow \{0,1\}^{|r|} : \langle\!|T\{r/H(e)\}|\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, r, \boldsymbol{L}_{\vec{H}})\right]\end{aligned} \tag{10}$$

We now provide justifications for the (in)equalities between probabilities.

(1): Holds because we only add ghost state and ϕ does not refer to it.

(2): Follows from Shoup's Fundamental Lemma [31] since $\Pr[G_1 : \psi] = \Pr[G_2 : \psi]$ and for any event ϕ, $\Pr[G_1 : \phi \wedge \neg\psi] = \Pr[G_2 : \phi \wedge \neg\psi]$.

(3): Follows from Shoup's Fundamental Lemma [31] since $\Pr[G_2 : \xi] = \Pr[G_3' : \xi]$ and for any event ϕ, $\Pr[G_2 : \phi \wedge \neg\xi] = \Pr[G_3' : \phi \wedge \neg\xi]$.

(4): Follows from the provided observations that the test can only succeed in the second phase and that if the test is true, then it remains true if additional elements are added to the lists $\boldsymbol{L}_H$.

(5) and (6): Hold because G_3' is equivalent to $\mathsf{CCA}(c^\star, T \wedge \mathsf{Ask}(H, e) \rhd t, \mathcal{R}, \vec{q}, \mathcal{A})$ except for the ghost state introduced in the first step which is not referenced by the events.

(7): Follows from the union bound and because i is sampled uniformly and is independent of the remaining memory of G_2.

(8): The first equality follows from the fact that ψ_i only refers to $\vec{pk}$, $\vec{sk}$ and the values $hist_{\boldsymbol{L}_H}[i]$, $hist_c[i]$, and $hist_{m_b}[i]$ of the history variables together with the observation that G_2 and G_3 behave identically until G_3 halts in the i-th query since both games do not change the values required to evaluate ψ_i. The second equality follows from $hist_c[i] = ci$, $hist_{\vec{\boldsymbol{L}}_H}[i] = \boldsymbol{L}_{\vec{H}}$ and $hist_{m_b}[i] = m_b$ in G_3 and the observation about the definedness of m_b.

(9): The equalities follow from Lemma 13, Lemma 11 and the definition of the procedure H for an eager random oracle.

(10): The equality follows from inlining the definition of the lazy random oracle procedure H and the observation that ρ implies that $(\!|e|\!)^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci) \notin \mathsf{dom}(\boldsymbol{H})$ when xi is computed. □

Theorem 28 (Soundness of CCA logic). *If ∇ is a derivation of $\vDash_{1/2} (c^\star, D) : \mathsf{Guess}$ under Γ, then*

$$\mathbf{Adv}_{\Pi}^{\mathsf{CCA}}(t_{\mathcal{A}}, \vec{q}) \leq 2\ \mathcal{B}_{(t_A, \vec{q})}(\nabla) - 1$$

Moreover, this bound is negligible if $t_{\mathcal{A}}$, all q_H in $\vec{q}$, and $q_{\mathcal{D}}$ are polynomial in the security parameter, and all assumptions in Γ hold.

Proof. From Lemma 27, for some CCA adversary $\mathcal{A}$,

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\mathsf{CCA}}(t_{\mathcal{A}}, \vec{q}) &= 2\Pr\left[\mathsf{CCA}(c^\star, \mathsf{Guess}, D, \mathcal{R}, \vec{q}, \mathcal{A}) : b = \bar{b}\right] - 1 \\ &= 2\Pr\left[\mathsf{CCA}(c^\star, \mathsf{Guess}, D, \mathcal{R}, \vec{q}, \mathcal{A}) : (\!|\mathsf{Guess}|\!)\right] - 1 \\ &\leq 2\,\mathcal{B}_{(t_A, \vec{q})}(\nabla) - 1\end{aligned}$$

The derivation ∇ must contain exactly one application of rule [Ind]. This is so because no branching rule contains in the premise more than one judgment with 1/2 as probability tag. Moreover, for a derivation ∇' that does not contain an application of [Ind], the bound $\mathcal{B}_{(t_A, \vec{q})}(\nabla')$ is negligible in the security parameter if $t_{\mathcal{A}}$, all q_H in $\vec{q}$, and $q_{\mathcal{D}}$ are polynomial in the security parameter. This can be proved by induction on the structure of ∇':

Case [Opt], [Perm], [Split], [Merge], [Sub], [$\mathsf{Fail}_{1,2}$], [Find], [Conv], **and** [$\mathsf{PAnd}_{1,2}$] Follow from the inductive hypothesis.

Case [$\mathsf{Eqs}(\ell)$], [$\mathsf{PEqs}(\ell)$] Since ℓ is interpreted as a non-constant polynomial in the security parameter, the bound $2^{-\ell}$ is negligible.

Case (extended) [$\mathsf{Rnd}(\ell, H)$] **and** [$\mathsf{PRnd}(\ell, H)$] Since ℓ is interpreted as a non-constant polynomial, and all q_H in $\vec{q}$ are polynomial in the security parameter, the bound $q_{\vec{H}} \times 2^{-\ell}$ is negligible.

```
$G_1 \overset{\text{def}}{=} G_0$ where
  $\mathcal{D}(c) =$
    foreach $H \in \mathcal{H}$ do $hist_{\boldsymbol{L}_H}[n_{\mathcal{D}}] \leftarrow \boldsymbol{L}_H$;
    $hist_c[n_{\mathcal{D}}] \leftarrow c$;
    $hist_{m_b}[n_{\mathcal{D}}] \leftarrow m_b$;
    if $\llparenthesis T \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}})$ then
      return $\llparenthesis t \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c)$
    return $\bot$
  $Main =$
    $i \xleftarrow{\$} \{0, \ldots, q_{\mathcal{D}} - 1\}$;
    foreach $H \in \mathcal{H}$ do $hist_{\boldsymbol{L}_H} \leftarrow \emptyset$;
    $hist_c \leftarrow \emptyset$;
    $hist_{m_b} \leftarrow \emptyset$;
    $G_0.Main$
```

```
$G_2 \overset{\text{def}}{=} G_1$ where
  $\mathcal{D}(c) =$
    foreach $H \in \mathcal{H}$ do $hist_{\boldsymbol{L}_H}[n_{\mathcal{D}}] \leftarrow \boldsymbol{L}_H$;
    $hist_c[n_{\mathcal{D}}] \leftarrow c$;
    $hist_{m_b}[n_{\mathcal{D}}] \leftarrow m_b$;
    if $\left( \begin{array}{l} \llparenthesis \mathsf{Ask}(H, e) \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}}) \vee \\ \llparenthesis e = e^\star \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, m_b) \end{array} \right)$ then
      if $\llparenthesis T \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}})$ then
        return $\llparenthesis t \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c)$
    return $\bot$
```

```
$G_3' \overset{\text{def}}{=} G_2$ where
  $\mathcal{D}(c) =$
    foreach $H \in \mathcal{H}$ do $hist_{\boldsymbol{L}_H}[n_{\mathcal{D}}] \leftarrow \boldsymbol{L}_H$;
    $hist_c[n_{\mathcal{D}}] \leftarrow c$;
    $hist_{m_b}[n_{\mathcal{D}}] \leftarrow m_b$;
    if $\llparenthesis T \wedge \mathsf{Ask}(H, e) \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}})$ then
      return $\llparenthesis t \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c)$
    else return $\bot$
```

```
$G_3 \overset{\text{def}}{=} G_2$ where
  $\mathcal{D}(c) =$
    foreach $H \in \mathcal{H}$ do $hist_{\boldsymbol{L}_H}[n_{\mathcal{D}}] \leftarrow \boldsymbol{L}_H$;
    $hist_c[n_{\mathcal{D}}] \leftarrow c$;
    $hist_{m_b}[n_{\mathcal{D}}] \leftarrow m_b$;
    if $n_{\mathcal{D}} = i$ then $ci \leftarrow c$; halt  // $(*)$
      if $\llparenthesis T \wedge \mathsf{Ask}(H, e) \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}})$ then
        return $\llparenthesis t \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c)$
      else return $\bot$
```

$$\psi \overset{\text{def}}{=} \exists i.\ 0 \le i < q_{\mathcal{D}} \wedge$$
$$(\text{let } c = hist_c[i] \text{ and } \boldsymbol{L}_{\vec{H}} = hist_{\vec{\boldsymbol{L}}_H}[i] \text{ and } v = \llparenthesis e \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c) \text{ in}$$
$$\llparenthesis T \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}}) \wedge v \notin \boldsymbol{L}_H \wedge (hist_{m_b}[i] = \bot \vee v \neq \llparenthesis e^\star \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)))$$

$$\xi \overset{\text{def}}{=} \exists i.\ 0 \le i < q_{\mathcal{D}} \wedge (\text{in } \llparenthesis T \wedge e = e^\star \rrparenthesis^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, hist_c[i], hist_{\vec{\boldsymbol{L}}_H}[i], hist_{m_b}[i]))$$

$(*)$: Note that we could also implement halt by prefixing *Main* with *halted* $\leftarrow$ false, using *halted* $\leftarrow$ true in place of halt, and replacing all commands c with if $\neg$ *halted* then c.

Figure 10: Games and events used in proof of [Bad] rule. See Figure 11 for remaining definitions.

```
$G_4 \overset{\text{def}}{=} G_3$ where
  $Main =$
    $i \xleftarrow{\$} \{0, \dots, q_{\mathcal{D}} - 1\}$;
    $x^\star \leftarrow \perp$;
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \mathsf{nil}$;
    $n_{\mathcal{D}} \leftarrow 0$;
    foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
    foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
    foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
    $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_{\mathcal{A}}, \mathcal{D}_1}(\vec{pk})$;
    $b \xleftarrow{\$} \{0,1\}$;
    $x^\star \leftarrow H(\langle\!\lbrack e^\star \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b))$;
    $c^\star \leftarrow \langle\!\lbrack c^\star\{x^\star/H(e^\star)\} \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$;
    $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_{\mathcal{A}}, \mathcal{D}_2}(c^\star, \sigma)$;
    $xi \leftarrow H(\langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci))$
  $\mathcal{D}(c) =$
    if $n_{\mathcal{D}} = i$ then $ci \leftarrow c$; halt
    else
      if $\left( \begin{array}{l} \langle\!\lbrack \mathsf{Ask}(H, e) \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}}) \vee \\ \langle\!\lbrack e = e^\star \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, m_b) \end{array} \right)$ then
        $x \leftarrow H(\langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b))$;
        if $\langle\!\lbrack T\{x/H(e)\} \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}})$ then
          return $\langle\!\lbrack t\{x/H(e)\} \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c)$
    return $\perp$
```

```
$G_6 \overset{\text{def}}{=} G_5$ where
  $Main =$
    $i \xleftarrow{\$} \{0, \dots, q_{\mathcal{D}} - 1\}$;
    $x^\star \leftarrow \perp$;
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \mathsf{nil}$;
    $n_{\mathcal{D}} \leftarrow 0$;
    foreach $\mathsf{f} \in \mathcal{F}$ do $(pk_f, sk_f) \leftarrow \mathcal{KG}_f$;
    foreach $H \in \mathcal{H}$ do $\boldsymbol{L}_H \leftarrow \mathsf{nil}$; $\mathsf{init}_H$;
    foreach $r \in \mathcal{R}$ do $r \xleftarrow{\$} \{0,1\}^{|r|}$;
    $(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1^{\vec{H}_{\mathcal{A}}, \mathcal{D}_1}(\vec{pk})$;
    $b \xleftarrow{\$} \{0,1\}$;
    $x^\star \leftarrow H(\langle\!\lbrack e^\star \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b))$;
    $c^\star \leftarrow \langle\!\lbrack c^\star\{x^\star/H(e^\star)\} \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$;
    $\bar{b} \leftarrow \mathcal{A}_2^{\vec{H}_{\mathcal{A}}, \mathcal{D}_2}(c^\star, \sigma)$;
    $xi \xleftarrow{\$} \{0,1\}^{|\mathsf{ran}(H)|}$
```

$$\psi_i \overset{\text{def}}{=} (\text{let } c = hist_c[i] \text{ and } \boldsymbol{L}_{\vec{H}} = hist_{\vec{\boldsymbol{L}}_H}[i] \text{ and } v = \langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c) \text{ in } \\ \langle\!\lbrack T \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, c, \boldsymbol{L}_{\vec{H}}) \wedge v \notin \boldsymbol{L}_H \wedge (hist_{m_b}[i] = \perp \vee v \neq \langle\!\lbrack e^\star \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)))$$

$$\psi' \overset{\text{def}}{=} \langle\!\lbrack T \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci, \boldsymbol{L}_{\vec{H}}) \wedge \langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci) \notin \boldsymbol{L}_H \wedge \langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci) = \langle\!\lbrack e^\star \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b)$$

$$\rho \overset{\text{def}}{=} \langle\!\lbrack T\{xi/H(e)\} \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci, xi, \boldsymbol{L}_{\vec{H}}) \wedge \langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci) \notin \boldsymbol{L}_H \wedge \langle\!\lbrack e \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, ci) \neq \langle\!\lbrack e^\star \rbrack\!\rangle^{\vec{\boldsymbol{H}}}(\vec{pk}, \vec{sk}, m_b))$$

Figure 11: Games and events used in proof of [Bad] rule.

Case (extended) $[\mathsf{OW}_k^\ell(\mathsf{f}, \vec{H})]$ Observe first that since $t_{\mathcal{A}}$ and all q_H in $\vec{q}$ are polynomial in the security parameter, $t_{\mathcal{I}}$ is also polynomial. Hence, since $(\mathsf{f}, k, \ell) \in \Gamma$, the bound $\mathbf{Succ}_{\Theta_\mathsf{f}}^{\text{s-pd-OW}}(k, \ell, q_{\vec{H}}, t_{\mathcal{I}})$ is negligible in the security parameter.

Case [False] Trivial.

Case [Bad] Follows from inductive hypothesis and the fact that $q_{\mathcal{D}}$ is polynomial in the security parameter.

Case $[\mathsf{Pub}(F, T, t)]$ Observe first that since $t_{\mathcal{A}}$, all q_H in $\vec{q}$, and $q_{\mathcal{D}}$ are polynomial in the security parameter, and T, t do not contain symbols of the form f^{-1}, $t_{\mathcal{S}}$ is also polynomial. Similarly, all q_H in $\vec{q}^{\mathcal{S}}$ are polynomial, because the derivation does not depend on the security parameter and $\mathsf{Q}_i(T), \mathsf{Q}_i(t)$ are constants.

Therefore, $\mathcal{B}_{(t_A, \vec{q})}(\nabla)$ is of the form $1/2 + \nu$ for some negligible function ν, and $2\,\mathcal{B}_{(t_A, \vec{q})}(\nabla) - 1 = \nu$ is negligible. $\square$

B Proof Generation

The proof generation mechanism offers an alternative method to validate the soundness of the logic. In the long term, we plan to provide a tight integration of ZooCrypt and EasyCrypt, in particular as a means to enhance automation in the latter.

The proof generation mechanism takes as input a valid derivation in the CPA logic and outputs an EasyCrypt proof script that can be verified independently. Generation is done in four steps:

1. build a context that declares all size variables, operator, constants and global variables required in the different games of the proof;
2. build the sequence of games, including the code of the inverters for leaves in the proof tree corresponding to applications of rule [OW];
3. output judgments in the relational logic of EasyCrypt to justify all branches in the derivation tree, inferring the necessary probability inequalities from them;
4. prove the concluding claim by combining all previous derived inequalities.

Currently, the proof generation mechanism is restricted to derivations that do not use the rule $[\mathsf{Fail}_2]$—this restricted class of derivations accounts for most examples in the literature. Extending the mechanism to the full CPA logic poses no difficulty.

Extending the proof generation mechanism to valid derivations of the CCA logic is less direct. It requires providing proof scripts to validate all instances of the [Bad] rule. We have written an EasyCrypt proof that validates the first application of the [Bad] rule in the proof of OAEP—the proof essentially follows the soundness argument given in the previous section. The proof could be used as a template for other cases; however, we believe that a better alternative is to adapt the proof generation mechanism to the forthcoming release of EasyCrypt and use its module and instantiation mechanisms to generate compact proof scripts.