Lectures on NIZKs: A Concrete Security Treatment

Mihir Bellare

January 14, 2020

Abstract

We initiate a concrete-security treatment of NIZKs. We start with definitions conducive to such a treatment. We give some basic general results on, and relations between, these notions. We then look at some NIZK systems in the literature and give concrete (as opposed to asymptotic) results about whatever properties they possess. We explore some basic applications of NIZKs, such as digital signatures, to give concrete security reductions. We define dual-mode proof systems as a way to formalize ideas underlying some NIZKs in the literature. This concrete security treatment of NIZKs is motivated by emerging applications, where it serves to help determine, and also reduce, parameters for a given level of security, leading to security-preserving efficiency gains. These notes were created as part of a course, taught in Winter 2020, on NIZKs. They represent a work in progress intended as a starting point for a future paper, not a finished product.

1 Department of Computer Science & Engineering, University of California San Diego, USA. Email: mihir@eng.ucsd.edu. URL: cseweb.ucsd.edu/~mihir/. Supported in part by NSF grant CNS-1228890, NSF grant CNS-1526801, ERC Project ERCC FP7/615074 and a gift from Microsoft corporation.
## Contents

1 Introduction 3

2 Asymptotic, concrete and blended security: An example 3
   2.1 Asymptotic ................................................................. 4
   2.2 Concrete ................................................................. 4
   2.3 Blended ................................................................. 6
   2.4 Discussion ............................................................. 7

3 Preliminaries 8

4 NIZK definitions and basic relations 12
   4.1 Claim validators and proof systems ................................. 13
   4.2 ZK and WI .............................................................. 13
   4.3 Basic soundness and extractability: SND1 and XT1 ............ 16
   4.4 Simulation soundness and extractability: SND2, XT2 ........... 16
   4.5 Joint notions: ZKSND, ZKXT ....................................... 17
   4.6 Dual-mode systems .................................................. 18
   4.7 Relations ............................................................. 19

5 Groups and bilinear maps 22

6 NIZKs from SUBD 25

7 Signatures from NIZKs 32
   7.1 Signature definitions .................................................. 32
   7.2 Construction from ZK+XT2 ........................................... 33
   7.3 Construction from ZK+XT1 ........................................... 36

References 36
1 Introduction

Non-interactive zero-knowledge (NIZK) systems are seeing increasing usage and application, making efficiency a target. But efficiency decoupled from security becomes nonsensical. The proper perspective is that we have some desired level of security (for example, 256 bits), for some set of goals (for example, soundness and zero-knowledge) and then want to minimize cost subject to staying at this level of security. To perform a design task like this in a rigorous way, we need quantitative metrics and results for NIZK security. This work initiates a treatment of NIZKs that provides this.

Zero-knowledge was introduced by [GMR89]. NIZKs were introduced by [BFM88, BDSMP91]. (The latter paper is the definitive reference, fixing the many bugs in the former.) Many constructions [FLS99, BY96, KP98] and applications [BG90, DDN00] followed. The treatment here was asymptotic. Security (for whatever notion) meant that polynomial-time adversaries have a success probability that is a negligible function of the security parameter. Such a treatment does not facilitate determining what setting of parameters provides some desired numeric level of security.

Applications have grown, fueling a search for efficient NIZKs. The Groth-Sahai framework [GS08] is widely utilized. We are seeing both new protocols and their implementation [GS08, Gro10, BCTV14, EG14]. Structure-preserving cryptography [AFG10, AGOT14, Gro15] was developed to allow these NIZKs to be used for efficient applications.

In these notes, we initiate a concrete security treatment of NIZKs. We return to the basic definitions and cast them via games and adversary advantage functions. We give quantitative results about relations between notions. We then cast some existing proof systems in this framework. And we look at the concrete security of reductions for applications that use NIZKs.

These notes are a work in progress, meant for a class I am teaching at UCSD in Winter 2020. They are in constant flux. Not only may things be added, but things can change. There will be mistakes, big and small.

In CSE207, the treatment was concrete. Students entering with only that background may not know what is an asymptotic treatment. No matter. For you, the current treatment of NIZKs may seem more natural.

There is some novelty in the definitions given here, as discussed further in Section 4. For example, we give joint definitions of ZK + soundness/extractability, allow multiple verification/extraction attempts, and formulate dual-mode NIZKs.

The notes include explicit Exercises. But they also include implicit ones, indicated sometimes just by the question “why?” somewhere in the text, prompting the reader to do the exercise of answering the question.

Comments, corrections, feedback, thoughts and suggestions, whether technical, historical or opinionated, are welcome.

2 Asymptotic, concrete and blended security: An example

This section introduces the reader to the asymptotic, concrete and blended settings for cryptographic definitions and proofs via a simple example. We define the OW (one-wayness) and PR (pseudo-randomness) security of a function, and show that the latter (under some conditions)
implies the former.

2.1 Asymptotic

A function \( \nu: \mathbb{N} \to \mathbb{N} \) is \textit{negligible} if for every positive polynomial \( p: \mathbb{N} \to \mathbb{R} \) there is a \( \lambda_p \in \mathbb{N} \) such that \( \nu(\lambda) \leq 1/p(\lambda) \) for all \( \lambda \geq \lambda_p \). “PT” stands for “polynomial time.” By \( 1^\lambda \) we denote the unary representation of the integer security parameter \( \lambda \in \mathbb{N} \).

A generator is a PT deterministic algorithm \( G \) that takes \( 1^\lambda \) and an input \( X \in \{0,1\}^{G,\text{il}(\lambda)} \) to return an output \( G(1^\lambda, X) \in \{0,1\}^{G,\text{ol}(\lambda)} \). Here \( G, \text{il}, G, \text{ol}: \mathbb{N} \to \mathbb{N} \) are PT deterministic functions determining the input and output lengths. We say that \( G \) is OW-secure (OW stands for “One-Way”) if for every PT adversary \( I \) the function

\[
\Pr \left[ G(1^\lambda, X') = Y : X \leftarrow \{0,1\}^{G,\text{il}(\lambda)} ; Y \leftarrow G(1^\lambda, X) ; X' \leftarrow I(1^\lambda, Y) \right] = \frac{1}{2}
\]

is negligible. We say that \( G \) is PR-secure (PR for “Pseudo-Random”) if for every PT adversary \( D \) the function

\[
\Pr \left[ D(1^\lambda, Y_b) = b : b \leftarrow \{0,1\} ; X \leftarrow \{0,1\}^{G,\text{il}(\lambda)} ; Y_1 \leftarrow G(1^\lambda, X) ; Y_b \leftarrow \{0,1\}^{G,\text{ol}(\lambda)} \right] - \frac{1}{2}
\]

is negligible.

\textbf{Theorem 2.1} Let \( G \) be a generator such that the function \( \ell(\lambda) = 2^{G,\text{il}(\lambda)} - G,\text{ol}(\lambda) \) is negligible. Assume \( G \) is PR-secure. Then \( G \) is OW-secure.

\textbf{Proof of Theorem 2.1:} Suppose towards a contradiction that \( G \) is not OW-secure. Then there exists a PT adversary \( I \) and a positive polynomial \( p_I: \mathbb{N} \to \mathbb{R} \) such that

\[
\Pr \left[ G(1^\lambda, X') = Y : X \leftarrow \{0,1\}^{G,\text{il}(\lambda)} ; Y \leftarrow G(1^\lambda, X) ; X' \leftarrow I(1^\lambda, Y) \right] \geq \frac{1}{p_I(\lambda)} \tag{1}
\]

for infinitely many \( \lambda \in \mathbb{N} \). We show that there exists a PT adversary \( D \) and a polynomial \( p_D: \mathbb{N} \to \mathbb{R} \) such that

\[
\Pr \left[ D(1^\lambda, Y_b) = b : b \leftarrow \{0,1\} ; X \leftarrow \{0,1\}^{G,\text{il}(\lambda)} ; Y_1 \leftarrow G(1^\lambda, X) ; Y_b \leftarrow \{0,1\}^{G,\text{ol}(\lambda)} \right]
\]

\[
\geq \frac{1}{2} + \frac{1}{p_D(\lambda)}
\]

for infinitely many \( \lambda \in \mathbb{N} \). This contradicts the assumption that \( G \) is PR-secure, establishing the theorem.

Adversary \( D(1^\lambda, Y) \) runs \( I(1^\lambda, Y) \) to get \( X' \). If \( G(1^\lambda, X') = Y \) then it outputs 1, else it outputs 0. If \( Y = G(X) \) for \( X \leftarrow \{0,1\}^{G,\text{il}(\lambda)} \) then the probability that \( D \) outputs 1 is at least the probability on the left of Equation (1). If \( Y \leftarrow \{0,1\}^{G,\text{ol}(\lambda)} \) then the probability that \( D \) outputs 1 is at most \( 2^{G,\text{il}(\lambda)} / 2^{G,\text{ol}(\lambda)} \) because an \( X' \) satisfying \( Y = G(X) \) exists with at most this probability. \( \blacksquare \)

2.2 Concrete

A generator is a deterministic algorithm \( G \) that takes an input \( X \in \{0,1\}^{G,\text{il}} \) to return an output \( G(X) \in \{0,1\}^{G,\text{ol}} \). Here \( G,\text{il}, G,\text{ol} \in \mathbb{N} \) are the input and output lengths. The ow-advantage of an
adversary $I$ is defined by
\[
\text{Adv}^\text{OW}_G(I) = \Pr[\text{G}^\text{OW}_G(I)]
\]
where game $\text{G}^\text{OW}_G$ is on the top left in Figure 1. The adversary must begin by calling oracle $\text{Init}$ (to get $Y$), and conclude by calling oracle $\text{Fin}$ with an argument $X' \in \{0, 1\}$. The notation $\Pr[\text{G}^\text{OW}_G(I)]$ refers to the probability that $\text{Fin}$ returns $\text{true}$ in the execution of the game with the adversary. The pr-advantage of an adversary $D$ is defined by
\[
\text{Adv}^\text{PR}_G(D) = 2\Pr[\text{G}^\text{PR}_G(D)] - 1
\]
where game $\text{G}^\text{PR}_G$ is on the top right in Figure 1. The conventions are similar.

**Theorem 2.2** Let $G$ be a generator. Given an adversary $I$ we can construct an adversary $D$ (shown explicitly at the bottom of Figure 1) such that
\[
\text{Adv}^\text{OW}_G(I) \leq \text{Adv}^\text{PR}_G(D) + \frac{1}{2^{G.\text{ol}-G.\text{il}}}.
\]

The running time of $D$ is about the same as that of $I$.

**Proof of Theorem 2.2:** Let $b$ be the challenge bit chosen at random in game $\text{G}^\text{PR}_G$, and $b'$ the bit queried by $D$ to $\text{G}^\text{PR}_G.\text{Fin}$. Then (appealing here to Lemma 3.2) we have
\[
\text{Adv}^\text{PR}_G(D) = \Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0]
\]
where the probabilities are in the execution of game $\text{G}^\text{PR}_G$ with adversary $D$. If $b = 1$ and $I$ succeeds,
every PT adversary $D$ for all $\lambda$

Putting the above together completes the proof.

The function $\text{Adv}^{\text{ow}}_D$ is defined by

determining the input and output lengths. The ow-advantage $\text{Adv}^{\text{ow}}_D = \text{Pr}[b' = 1 | b = 1]$.

Let $R = \{ G(X) : X \in \{0, 1\}^{G,\text{il}} \subseteq \{0, 1\}^{G,\text{ol}} \}$. If $b = 0$ then

$$\text{Pr}[b' = 1 | b = 0] = \text{Pr}[Y \in R] = \frac{|R|}{2^{G,\text{ol}}} \leq \frac{2^{G,\text{il}}}{2^{G,\text{ol}}}.$$ 

Putting the above together completes the proof. 

### 2.3 Blended

A generator is a PT deterministic algorithm $G$ that takes $1^\lambda$ and an input $X \in \{0, 1\}^{G,\text{il}(1^\lambda)}$ to return an output $G(1^\lambda, X) \in \{0, 1\}^{G,\text{ol}(1^\lambda)}$. Here $G,\text{il}, G,\text{ol}: \mathbb{N} \to \mathbb{N}$ are PT deterministic functions determining the input and output lengths. The ow-advantage $\text{Adv}^{\text{ow}}_{G,1}(\cdot): \mathbb{N} \to \mathbb{R}$ of an adversary $I$ is defined by

$$\text{Adv}^{\text{ow}}_{G,1}(\lambda) = \text{Pr}[G^{\text{ow}}_{G,1}(I)]$$

for all $\lambda \in \mathbb{N}$, where game $G^{\text{ow}}_{G,1}$ is on the top left in Figure 2. We say that $G$ is OW-secure if for every PT adversary $I$ the function $\text{Adv}^{\text{ow}}_{G,1}(\cdot)$ is negligible. The pr-advantage $\text{Adv}^{\text{pr}}_{G,1}(\cdot): \mathbb{N} \to \mathbb{R}$ of an adversary $D$ is defined by

$$\text{Adv}^{\text{pr}}_{G,1}(\lambda) = 2 \text{Pr}[G^{\text{pr}}_{G,1}(D)] - 1$$

for all $\lambda \in \mathbb{N}$, where game $G^{\text{pr}}_{G,1}$ is on the top right in Figure 2. We say that $G$ is PR-secure if for every PT adversary $D$ the function $\text{Adv}^{\text{pr}}_{G,1}(\cdot)$ is negligible.
Theorem 2.3 Let $G$ be a generator. Given an adversary $I$ we can construct an adversary $D$ (shown explicitly at the bottom of Figure 2) such that for all $\lambda \in \mathbb{N}$ we have

$$\text{Adv}_{G,I}^{ow}(\lambda) \leq \text{Adv}_{G,D}^{pr}(\lambda) + \frac{1}{2^{G\text{.ol}(1^{\lambda}) - G\text{.il}(1^{\lambda})}}.$$  \hspace{1cm} (3)

The running time of $D$ is about the same as that of $I$.

Proof of Theorem 2.3: Let $\lambda \in \mathbb{N}$. Let $b$ be the challenge bit chosen at random in game $G_{G,A}^{pr}$, and $b'$ the bit queried by $D$ to $G_{G,A}^{pr}$, FIN. Then (appealing here to Lemma 3.2) we have

$$\text{Adv}_{G,D}^{pr}(\lambda) = \Pr \left[ b' = 1 \mid b = 1 \right] - \Pr \left[ b' = 1 \mid b = 0 \right]$$

where the probabilities are in the execution of game $G_{G,A}^{pr}$ with adversary $D$. If $b = 1$ and $I$ succeeds, meaning returns an $X'$ such that $G(1^{\lambda}, X') = Y$, then $b'$ equals 1, so we have

$$\Pr \left[ b' = 1 \mid b = 1 \right] \geq \text{Adv}_{G,I}^{ow}(\lambda).$$

Let $R = \{ G(X) : X \in \{0,1\}^{G\text{.il}(1^{\lambda})} \} \subseteq \{0,1\}^{G\text{.ol}(1^{\lambda})}$. If $b = 0$ then

$$\Pr \left[ b' = 1 \mid b = 0 \right] \leq \Pr[Y \in R] = \frac{|R|}{2^{G\text{.ol}(1^{\lambda})}} \leq \frac{2^{G\text{.il}(1^{\lambda})}}{2^{G\text{.ol}(1^{\lambda})}}.$$

Putting the above together completes the proof.

Note that Theorem 2.1 is a corollary of Theorem 2.3, but not, at least formally, of Theorem 2.2.

2.4 Discussion

Schemes (here, $G$) have a different syntax in the asymptotic and concrete settings, taking input a security parameter (in unary) in the former, but not in the latter. In the concrete treatment, the security parameter does not exist.

Some real-world primitives, like AES or SHA256, do not have a security parameter and would not fit the asymptotic setting. The concrete setting covers them.

In the asymptotic setting, what it means for a scheme (here $G$) to be secure (here in the PR or OW sense) is formally well defined. In the concrete setting, it is not: we define the security metric via the adversary advantage function, but stop short of formally defining what it means for the scheme to be “secure” under this metric. When we say $G$ is PR-secure, it is an informal statement, to be interpreted as $\text{Adv}_G^{pr}(D)$ is “small” for all adversaries $D$ whose resources are “practical.” This is generally not a difficulty with simple notions like the ones in this example. But it makes more difficult the conceptual understanding of more complex notions like ZK which involve a particular quantification over different objects (adversary, simulator and so on) that, in the concrete setting, are all just parameters.

The asymptotic setting does not (usually) define any explicit advantage function. This makes it difficult (but not impossible) to make quantitative statements about the relationships between these advantages and (in my view) makes it harder than in the concrete setting to write precise proofs, as can be seen by comparing the proofs of Theorems 2.1 and 2.2. Of course, one could define advantage functions, and this is exactly what the blended setting does, but historically, and in canonical treatments [Gol01], this does not seem to be done.
The asymptotic setting does not (usually) use games, instead expressing adversary success probabilities directly. It tends to present adversaries in text rather than in pseudocode.

Theorems are formal statements in both settings. But the ones in the concrete setting explicitly state relations between adversary advantages and resources, and often even point to an explicit pseudocode adversary construction as the final determinant of its resource utilization. The running-time relation, however, tends to be a bit fuzzy for lack of a precise computational model, and we see statements, like in Theorem 2.2, about one running time being “about the same” as the other. Subtleties of which to be aware include code-size as an adversary resource and the consequences of the inherent non-uniformity of the setting [BL13].

Proofs in the asymptotic setting have tended, historically, to proceed by contradiction, with the template illustrated in the proof of Theorem 2.1, in contrast to proofs in the concrete setting, which proceed in the direct way illustrated by the proof of Theorem 2.2.

The notions we have considered here are simple. In asymptotic definitions for more complex notions, one can see many different parameters that must be appropriately quantified relative to something defined as being negligible. Asymptotic definitions of this type in the literature can be ambiguous, and even incorrect, as illustrated for example by [BH15]. The blended setting is a good remedy, forcing one to pin down the advantage function asked to be negligible.

The blended setting tries to be the “best of the two worlds.” It has a security parameter, and does define adversary advantages, but, rather than numbers, these are now functions, of the security parameter. What it means for a scheme to be secure is formally well-defined, and identical to the asymptotic setting. However, theorems are able to state the relations between adversary advantages and resources as in the concrete setting. Subtleties as mentioned above are largely avoided. (It isn’t perfect; for example, it does not directly capture AES and SHA256. But there are ways around this.)

The blended setting would be my choice for treating NIZKS, and what I would use in a paper on the subject. But for simplicity, these notes use the concrete setting.

3 Preliminaries

Notation. If \( w \) is a vector then \(|w|\) is its length (the number of its coordinates) and \( w[i] \) is its \( i \)-th coordinate. Strings are identified with vectors over \( \{0, 1\} \), so that \(|Z|\) denotes the length of a string \( Z \) and \( Z[i] \) denotes its \( i \)-th bit. By \( \varepsilon \) we denote the empty string or vector. By \( x\|y \) we denote the concatenation of strings \( x, y \). If \( x, y \) are equal-length strings then \( x \oplus y \) denotes their bitwise xor. If \( S \) is a finite set, then \(|S|\) denotes it size. We say that a set \( S \) is length-closed if, for any \( x \in S \) it is the case that \( \{0, 1\}^{\|x\|} \subseteq S \). (This will be a requirement for message spaces.)

If \( X \) is a finite set, we let \( x \leftarrow X \) denote picking an element of \( X \) uniformly at random and assigning it to \( x \). Algorithms may be randomized unless otherwise indicated. If \( A \) is an algorithm, we let \( y \leftarrow A^{O_1;\ldots;\omega}(x_1, \ldots) \) denote running \( A \) on inputs \( x_1, \ldots \) and coins \( \omega \), with oracle access to \( O_1, \ldots, \) and assigning the output to \( y \). By \( y \leftarrow A^{O_1;\ldots}(x_1, \ldots) \) we denote picking \( \omega \) at random and letting \( y \leftarrow A^{O_1;\ldots;\omega}(x_1, \ldots) \). We let \([A^{O_1;\ldots;\omega}(x_1, \ldots)]\) denote the set of all possible outputs of \( A \) when run on inputs \( x_1, \ldots \) and with oracle access to \( O_1, \ldots \). An adversary is an algorithm. Running time is worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. We use \( \bot \) (bot) as a special symbol to denote rejection, and it is assumed to not be in \( \{0, 1\}^* \).
Games. We use the code-based game-playing framework of BR [BR06]. A game G (see Figure 4 for examples) starts with an optional Init procedure, followed by a non-negative number of additional procedures called oracles, and ends with a Fin procedure. Execution of adversary A with game G consists of running A with oracle access to the game procedures, with the restrictions that A’s first call must be to Init (if present), its last call must be to Fin, and it can call these procedures at most once. The output of the execution is the output of Fin. By Pr[G(A) ⇒ y] we denote the probability that the execution of game G with adversary A results in this output being y, and write just Pr[G(A)] when y = true. (Meaning Pr[G(A)] is the probability that the execution of game G with adversary A results in the output of the execution being the boolean true.)

Note that our adversaries have no output. The role of what in other treatments is the adversary output is, for us, played by the query to Fin.

Different games may have procedures (oracles) with the same names. If we need to disambiguate, we may write G.O to refer to oracle O of game G.

In games, integer variables, set variables boolean variables and string variables are assumed initialized, respectively, to 0, the empty set ∅, the boolean false and ⊥.

The running time of an adversary executing with a game excludes the time taken by game procedures to compute answers to queries.

Game-playing lemmas. A flag is a boolean variable. Recall that any flag is initialized to false. If game G contains a flag bad then “G(A) sets bad” refers to the event that bad is set to true at some point in the execution of G with A. (As a clarification, this does not mean that bad is necessarily true when the game terminates.) We say that games G₀, G₁ are identical-until-bad if bad is a flag in both games and the games differ only in code following a statement bad ← true. The following is the Fundamental Lemma of Game Playing from [BR06].

Lemma 3.1 Let G₀, G₁ be identical-until-bad games, and A an adversary. Then for all b ∈ {0, 1} and all y we have

\[ \Pr[G_0(A) ⇒ y] − \Pr[G_1(A) ⇒ y] ≤ \Pr[G_b(A) sets \text{ bad}] . \]

Also

\[ \Pr[G_0(A) sets \text{ bad}] = \Pr[G_1(A) sets \text{ bad}] \]
\[ \Pr[G_0(A) ⇒ y \land \text{Gd}] = \Pr[G_1(A) ⇒ y \land \text{Gd}] , \]

where Gd denotes the event that bad is never set to true.

Above, for b ∈ {0, 1}, event Gd in the execution of G_b with A is the complement of the event “G_b(A) sets bad”.

A decision problem is one where the adversary’s task is to guess a challenge bit. One can formulate such a problem either via single game with the advantage defined as twice the probability of guessing the challenge bit minus one, or via two games, with the advantage defined as the difference in probabilities that the adversary’s guess is 1, and these two formulations are equivalent. Since this equivalence is used often and for many problems, we try here to formalize a statement about it.

Let G be a game. We say that it is a decision game if the following hold. There is a boolean variable b (called the challenge bit) such that G.Init includes the statement b ← s {0, 1}. This is the only code in the game that changes the value of b, so that the game in particular has no other
statement assigning a value to \( b \). The input to \( G.\text{Fin} \) includes a bit \( b' \) (call the guess bit), and \( G.\text{Fin} \), via a statement “return \((b' = b)\)” that is the only one in this procedure containing the “return” instruction, returns the boolean \((b' = b)\). This procedure has no statement changing the value of, or assigning a value to, \( b' \). By \( G[0] \) we denote the \( G \) with the following modifications. The variable \( b \) set to 0. This means the statement \( b \leftarrow \{0, 1\} \) is removed from \( \text{Init} \), and, for any reference to variable \( b \) in \( G \), game \( G[0] \) uses the value 0. Oracle \( G[0].\text{Fin} \) takes the same inputs as \( G.\text{Fin} \), including \( b' \), but replaces the “return \((b' = b)\)” statement by “return \((b' = 1)\),” meaning returns \( \text{true} \) iff \( b' \) is the bit 1. Game \( G[1] \) is defined correspondingly.

We warn that the above definitions are not entirely rigorous in the absence of a precise programming language [BR06]. However, we will apply the Lemma below only for explicitly-specified games \( G \) that we write in our definitions and proofs, and in these cases the conditions, and objects referred to, will be clear enough. But we warn that we are not yet at the stage of being able to formally and rigorously treat a game as an abstract mathematical object, and recommend caution in trying to doing this.

The following says that for decision games, the two ways of defining adversary advantage coincide.

**Lemma 3.2** Let \( G \) be a decision game, and \( A \) an adversary.

\[
2 \Pr[G(A)] - 1 = \Pr[G[1](A)] - \Pr[G[0](A)].
\]

**Proof of Lemma 3.2:** Let \( b \) be the challenge bit and \( b' \) the guess bit in \( G \), and consider the execution of \( G \) with \( A \). We have

\[
\Pr[b' = 1 \mid b = 1] = \Pr[G[1](A)] \tag{4}
\]

\[
\Pr[b' = 1 \mid b = 0] = \Pr[G[0](A)] \tag{5}
\]

Given this we have

\[
2 \Pr[G(A)] - 1 = 2 \Pr[b' = b] - 1 \\
= 2 \cdot \left( \Pr[b' = 1 \mid b = 1] \cdot \Pr[b = 1] + \Pr[b' = 0 \mid b = 0] \cdot \Pr[b = 0] \right) - 1 \\
= 2 \cdot \left( \Pr[b' = 1 \mid b = 1] \cdot \frac{1}{2} + \left( 1 - \Pr[b' = 1 \mid b = 0] \right) \cdot \frac{1}{2} \right) - 1 \\
= 2 \cdot \left( \Pr[b' = 1 \mid b = 1] \cdot \frac{1}{2} - \Pr[b' = 1 \mid b = 0] \cdot \frac{1}{2} + \frac{1}{2} \right) - 1 \\
= \Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0] \\
= \Pr[G[1](A)] - \Pr[G[0](A)],
\]

where the last equality uses Equations (4) and (5). \( \square \)

**Reductions.** Proofs give reductions that take a \( G_2 \)-adversary \( A_2 \) and specify (construct) a \( G_1 \)-adversary \( A_1 \) that runs \( A_2 \) as a subroutine, itself responding to oracle queries of \( A_2 \). Let \( \text{Init}, O_1, \ldots, O_{n_1}, \text{Fin} \) denote the oracles of \( G_1 \) and \( \text{Init}, O_{21}, \ldots, O_{2n_2}, \text{Fin} \) the oracles of \( G_2 \). Then we may write pseudocode of the form

\[
\text{Adversary } A_1
\]
We define PRF security for function family $F$ by the adversary to
be in $\Pr \left[ G_\text{prf}^b(A) \right]$ where the game is on the left in Figure 3. This assumes $F.D$ is finite. It is required that the $X'$ queried by the adversary to $\text{FIN}$ be in $F.D$.

We define PRF security for function family $F$ via the game $G_\text{prf}^b$ on the right in Figure 3. Here $b$ is the output gate and wire $C_g$ is the output wire. There is only one type of gate, which is a 2-input NAND.

Function $\text{Ev}$ evaluates a circuit. It takes a circuit $C$ and a string $x \in \{0, 1\}^{C.I2}$ giving assignments to the inputs of the circuit. It returns a vector $w$ such that $w[i]$ is the value of wire $i$. In particular, $w[C.I1 + C.Gt]$ is the circuit output:

<table>
<thead>
<tr>
<th>Algorithm $\text{Ev}(C, x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $i = 1, \ldots, C.In$ do $w[i] \leftarrow x[i]$</td>
</tr>
<tr>
<td>For $i = C.In + 1, \ldots, C.Gt$ do $w[i] \leftarrow \text{NAND}(C.I1(i), C.I2(i))$</td>
</tr>
<tr>
<td>Return $w$</td>
</tr>
</tbody>
</table>

Here $\text{Init}, O2_1, \ldots, O2_{n_2}, \text{Fin}$ are subroutines, given in the code of $A_1$, that are responsible for simulating the corresponding oracles for $A_2$ in $G_2$, and will invoke $A_1$'s oracles to do so. These subroutines can invoke oracles from $G_1$, which, to disambiguate, will the given the game name as prefix, so that for example subroutine $\text{Init}$ can call $G_1.\text{Init}$ and subroutine $\text{Fin}$ can call $G_1.\text{Fin}$.

We adopt the convention that if a simulation is trivial, meaning $O2_i(x)$ returns $O1_j(x)$, then, in the superscripts to $A_2$, we simply write $O1_j$ in place of $O2_i$, and do not give code for the simulated oracle.
is the challenge bit. The prf advantage of adversary $A$ is $\text{Adv}_{F}^{\text{prf}}(A) = 2 \Pr[G_{F}^{\text{prf}}(A)] - 1$.

**Commitment schemes.** A commitment scheme $CS$ specifies a parameter-generation algorithm $CS.P$ and a deterministic commitment algorithm $CS.C$: $\{0,1\}^{*} \times CS.K \times CS.M \rightarrow \{0,1\}^{*}$. The scheme is set up by generating parameters $cp \leftarrow CS.P$. Then via $c \leftarrow CS.C(cp, K, m)$, one generates a commitment to message $m$ with de-commitment key $K$.

## 4 NIZK definitions and basic relations

A proof system provides a way for one party (the prover) to prove some “claim” to another party (the verifier). A claim is defined via a claim statement, which is a string $x$, and a claim validator $CV$, which is an algorithm that, with the help of a witness string $w$, says whether or not the claim statement is valid. What we call a claim validator is, in the literature, often called the relation, the claim statement being the input or theorem.

We start by formalizing claim validators, the associated set of true claims, and the syntax of proof systems. We formalize security properties of proof systems in three clusters. First is basic properties: soundness, extractability and ZK. Second is simulation soundness and extractability. Third is joint notions where a single game asks simultaneously for ZK and either soundness or extractability.

A few elements here seem new. One is the joint definitions, which facilitate applications, allowing a few hybrid proof steps to be compressed into one. Another is allowing multiple verification (for soundness) or extraction (for extractability) queries via an oracle for this purpose. (Conventional treatments, in this language, correspond to allowing one query.) This does not change the notion asymptotically, but it changes the concrete security, and allows us to see how constructions differ with regard to the way adversary advantage grows as a function of the number of queries to these oracles. We also introduce dual-mode NIZKs as a way to formalize something implicit in the literature.

These are not the only definitions. The literature considers many more.
4.1 Claim validators and proof systems

Claim validators. A claim validator is a function \( \text{CV}: \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \rightarrow \{\text{true}, \text{false}\} \) that takes parameters pars, a claim statement \( x \) and a candidate witness \( w \) to return either true (saying \( w \) is a valid witness establishing the claim) or false (the witness fails to validate the claim).

Common examples arise from the relations underlying NP languages. For example, \( x \) could be a circuit, \( w \) an assignment to its variables, and \( \text{CV} \) returns true iff \( w \) satisfies \( x \). (In this example, \( \text{pars} = \varepsilon \).) For \( \text{pars} \in \{0,1\}^* \) we let \( \text{CV}(\text{pars}, x) = \{ w : \text{CV}(\text{pars}, x, w) = \text{true} \} \) be the witness set of \( x \).

A true-claim language is a function \( \text{TC}: \{0,1\}^* \rightarrow 2^{\{0,1\}^*} \), meaning it associates to any \( \text{pars} \in \{0,1\}^* \) a set \( \text{TC}(\text{pars}) \subseteq \{0,1\}^* \). A claim validator \( \text{CV} \) gives rise to the true-claim language \( \text{TrCl}_{\text{CV}} \) defined by \( \text{TrCl}_{\text{CV}}(\text{pars}) = \{ x : \text{CV}(\text{pars}, x) \neq \emptyset \} \) called the true-claim language associated to \( \text{CV} \). It associates to \( \text{pars} \) the set of valid (true) claim statements.

In usage of the proof system, the prover and verifier both know \( \text{pars}, x \). (This situation may be arrived at in many ways, including the prover supplying \( x \) while \( \text{pars} \) being trusted parameters.) The prover is claiming that \( x \in \text{TrCl}_{\text{CV}}(\text{pars}) \), and wants to establish this claim without revealing a witness \( w \) satisfying \( \text{CV}(\text{pars}, x, w) = \text{true} \) or, for that matter, anything else.

In the literature, \( \text{CV} \) often does not take \( \text{pars} \) as a separate input.

Proof systems. A proof system is the name of the syntax for the primitive that enables the production and verification of such proofs. Soundness, zero-knowledge and many other things will be security metrics for this primitive.

Proceeding, a proof system \( \Pi \) specifies the following algorithms:

- **CRS generation.** Via \( \text{crs} \leftarrow \Pi.C \), the crs-generation algorithm \( \Pi.C \) (takes no inputs and) returns an output \( \text{crs} \) called the common reference string.

- **Proof generation.** Via \( \text{pf} \leftarrow \Pi.P(\text{crs}, x, w) \) the proof generation algorithm \( \Pi.P \) takes \( \text{crs} \), a claim \( x \) and a witness \( w \) to produce a proof string. As this indicates the common reference string plays the role of the parameters.

- **Proof verification.** Via \( d \leftarrow \Pi.V(\text{crs}, x, \text{pf}) \) the proof verification algorithm \( \Pi.V \) produces a decision \( d \in \{\text{true}, \text{false}\} \) indicating whether or not it considers \( \text{pf} \) valid.

In some treatments, a proof system \( \Pi \) is defined as being for a claim validator \( \text{CV} \). Our syntax views \( \Pi \) independently of \( \text{CV} \). The claim validator shows up in defining attributes of \( \Pi \) below.

Completeness. We require perfect completeness, although this can be relaxed if necessary. We say that \( \Pi \) has (perfect) completeness for \( \text{CV} \) if \( \Pi.V(\text{crs}, x, \Pi.P(\text{crs}, x, w)) = \text{true} \) for all \( \text{crs} \in [\Pi.C] \), all \( x \in \text{TrCl}_{\text{CV}}(\text{crs}) \) and all \( w \in \text{CV}(\text{crs}, x) \).

4.2 ZK and WI

Zero knowledge. Zero knowledge (ZK) of \( \Pi \) for \( \text{CV} \) asks that there be a simulator that, given a true statement (meaning, one in \( \text{TrCl}_{\text{CV}}(\text{crs}) \)) can create for it a proof indistinguishable from one
This asks that, knowing Witness indistinguishability.

Again, a more fine-grained classification becomes possible. A requirement may be understood as asking that there is an “efficient” simulator $S$ or $\text{Adv}_{\Pi}$ such that $0$ for all $A$ (perfect), $\text{Adv}_{\Pi}^k$ or $\text{Adv}_{\Pi}^w$ is negligible for all $A$ (statistical) or $\text{Adv}_{\Pi}^k$ or $\text{Adv}_{\Pi}$ is negligible for all polynomial-time $A$ (computational). In the concrete setting, the requirement may be understood as asking that there is an “efficient” simulator $S$ such that $\text{Adv}_{\Pi}^k$ or $\text{Adv}_{\Pi}$ is negligible for all $A$ (perfect), is “small” for all $A$ (statistical) or is “small” for all practical $A$ (computational). Again, a more fine-grained classification becomes possible.

**Witness indistinguishability.** This asks that, knowing $x \in \text{TrCl}_{\Pi}(\text{crs})$ and knowing two
Adversary $A_{zk}$:

1. $\text{crs} \leftarrow G^{zk}_{\Pi, CV, S, \text{INIT}} ; d \leftrightarrow \{0, 1\} ; A^{\text{INIT, Pf, Fin}}_{wi}$

$\text{INIT()}$:
2. Return $\text{crs}$

$\text{Pr}(x, w_0, w_1)$:
3. If $((\text{CV} (\text{crs}, x, w_0) = \text{false}) \text{ or } (\text{CV} (\text{crs}, x, w_1) = \text{false}))$ then return $\bot$
4. $\text{pf} \leftarrow \Pi . \text{P}(\text{crs}, x, w_d) ; \text{Return pf}$

$\text{Fin}(d')$:
5. If $(d = d')$ then $b' \leftarrow 1$ else $b' \leftarrow 0$
6. $G^{zk}_{\Pi, CV, S, \text{Fin}}(b')$

Figure 5: Adversary for Proposition 4.1.

Witnesses $w_0, w_1 \in \text{CV}(\text{crs}, x)$, it is hard to tell under which of the two a proof has been computed. Consider game $G^{wi}_{\Pi, CV}$ specified in Figure 4. Let $\text{Adv}^{wi}_{\Pi, CV}(A) = 2 \Pr[G^{wi}_{\Pi, CV}(A)] - 1$.

ZK implies WI. The following says that ZK implies WI regardless of the running time of the simulator. (That the latter does not matter is reflected in the running time of $A_{zk}$ not depending on the running time of the algorithms of $S$.)

**Proposition 4.1** [ZK $\Rightarrow$ WI] Let CV be a claim validator, $\Pi$ a proof system, and $S$ a simulator. Let $A_{wi}$ be an adversary. Then we can construct adversary $A_{zk}$ (shown explicitly in Figure 5) such that

$$\text{Adv}^{wi}_{\Pi, CV}(A_{wi}) \leq 2 \cdot \text{Adv}^{zk}_{\Pi, CV, S}(A_{zk}) .$$

Adversary $A_{zk}$ makes the same number of Pf queries as $A_{wi}$ and its running time of $A_{zk}$ is about that of $A_{wi}$.

**Proof of Proposition 4.1:** We make use of Lemma 3.2. Let $b$ denote the challenge bit in the execution of $A_{zk}$ with game $G^{zk}_{\Pi, CV, S}$. Then

$$\Pr[b' = 1 \mid b = 1] = \frac{1}{2} + \frac{1}{2} \cdot \text{Adv}^{wi}_{\Pi, CV}(A_{wi})$$

$$\Pr[b' = 1 \mid b = 1] = \frac{1}{2} .$$

So

$$\text{Adv}^{zk}_{\Pi, CV, S}(A_{zk}) = \Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 1] = \frac{1}{2} \cdot \text{Adv}^{wi}_{\Pi, CV}(A_{wi}) .$$

Make sure you understand why the different equations are true. We omit the details.
4.3 Basic soundness and extractability: SND1 and XT1

SND1 SOUNDNESS. Soundness of \( \Pi \) is defined relative to a true-claim language \( TC \) which may then be set to \( TrCl_{CV} \) for some choice of \( CV \). It asks that it be hard to create a valid proof for \( x \not\in TC(crs) \). Our formalization is via game \( G^{snd1} \) in Figure 4. The adversary gets the common reference string \( crs \). Then, it can submit to \( Vf \) a statement \( x \) of its choice and a candidate proof \( pf \) for it, winning if \( x \not\in TC(crs) \) yet the verification algorithm accepts \( pf \) as valid. It may call the oracle multiple times, as often as it wants. We let \( Adv^{snd1}_{\Pi,TC}(A) = \Pr[G^{snd1}_{\Pi,TC}(A)] \) be its snd1-advantage.

The usual (asymptotic) definitions consider only one pair \( x, pf \) submitted by the adversary, which in our definition corresponds to limiting attention to adversaries that make a single \( Vf \) query. Proposition 4.9 shows, via the simple and expected hybrid argument, that an adversary making \( q \) queries has an advantage at most \( q \) times that of an adversary of comparable resources making just one query. In the asymptotic security mindset, this justifies not considering multiple queries. In the concrete security mindset, one goes further, asking, whether, for particular proof systems, there is a better reduction or result that avoids this factor \( q \) degradation. We will see that, for many proof systems, this is possible.

In the asymptotic setting, one speaks of soundness of \( \Pi \) being perfect (\( Adv^{snd}_{\Pi,TC}(A) = 0 \) for all \( A \)), statistical (\( Adv^{snd}_{\Pi,TC}(A) \) is negligible for all \( A \)) or computational (\( Adv^{snd}_{\Pi,TC}(A) \) is negligible for all polynomial-time \( A \)). In the concrete setting, perfect soundness is still well-defined. Statistical would be understood as \( Adv^{snd}_{\Pi,TC}(A) \) being “small” for all \( A \), and computational as it being “small” for all “practical” \( A \). But in the concrete setting, a more fine-grained differentiation of levels of soundness is possible, in which statistical and computational are ends of a spectrum with much in between.

XT1 EXTRACTABILITY. The notion of \( \Pi \) being a proof of knowledge [GMR89, BG93, DP92] for \( CV \) requires that whenever a (potentially cheating) prover, modeled as the adversary, is able to produce a valid proof, there is an extractor that, based on a trapdoor underlying the common reference string, can extract the witness from the information available to the adversary. Our formalization is via game \( G^{xt1} \) specified in Figure 4. It is parameterized by an extractor \( S \), an object that specifies algorithms \( S.C \) (the extraction-CRS generator) and \( S.X \) (the extraction witness-generator). Let \( Adv^{xt1}_{\Pi,CV,S}(A) = \Pr[G^{xt1}_{\Pi,CV,S}(A)] \) be the xt1-advantage of \( A \).

The asymptotic definition of \( \Pi \) being XT1-secure would be that there exists a polynomial time extractor \( S \) such that \( Adv^{xt1}_{\Pi,CV,S}(A) \) is negligible for all polynomial time adversaries \( A \). In the concrete setting, the requirement may be understood as asking that there is an “efficient” extractor \( S \) such that \( Adv^{xt1}_{\Pi,CV,S}(A) \) is “small” for all practical adversaries \( A \).

Exercise 4.2 Prove or disprove each of the following: (1) SND1 implies XT1 (2) XT1 implies SND1 (3) XT1+ZK (where the simulator is the same across the two) implies SND1.

4.4 Simulation soundness and extractability: SND2, XT2

SND2 SOUNDNESS. Simulation soundness [Sah99] requires that it is hard to create a valid proof for \( x \not\in TrCl_{CV}(pars) \) even after seeing simulated proofs on claims of the adversary’s choosing. Our formalization considers game \( G^{snd2}_{\Pi,CV,S} \) shown in Figure 6. Here \( S \), again, is a simulator, specifying
Exercise 4.3 Consider a definition of soundness, call it SND3, that is like SND2 except that the first instruction of line 3 is replaced with \( \text{pf} \leftarrow \Pi.P(\text{crs}, x, w) \). Explore the relation of this to SND1, paying attention to the concrete security.

XT2 extractability. Simulation-sound extractability requires that one be able to extract a witness underlying a valid proof even when the claim was created after seeing simulated proofs. Our formalization considers game \( G^{xt2}_{\Pi, CV, S} \) shown in Figure 6. Here \( S \), the simulation extractor, specifies algorithms \( S.C \) and \( S.P \). We let \( \text{Adv}^{xt2}_{\Pi, CV, S}(A) = \Pr[G^{xt2}_{\Pi, CV, S}(A)] \).

Exercise 4.4 Prove or disprove each of the following. (1) SND2+ZK implies SND1 (2) SND1+ZK implies SND2 (3) XT2+ZK implies XT1 (4) XT1+ZK implies XT2.

4.5 Joint notions: ZKSND, ZKXT

We give some new definitions where ZK and soundness/extractability are captured in a single game. This makes proofs of applications easier. Let \( \text{Adv}^{zksnd}_{\Pi, CV, S}(A) = 2 \Pr[G^{zksnd}_{\Pi, CV, S}(A)] − 1 \) and \( \text{Adv}^{zkxt}_{\Pi, CV, S}(A) = 2 \Pr[G^{zkxt}_{\Pi, CV, S}(A)] − 1 \) where the games are in Figure 7.

Exercise 4.5 Prove or disprove each of the following. (1) SND2+ZK implies ZKSND (2) ZKSND implies SND2+ZK (3) XT2+ZK implies ZKXT (4) ZKXT implies XT2+ZK.
Theorem 4.6 says that if one of the induced proof systems $\Pi_\mu$ satisfies ZK with respect to some simulator $S$, and the modes are indistinguishable, then the other induced proof system $\Pi_{1-\mu}$ also satisfies ZK with respect to the same $S$. The statement is in the concrete security framework. Given

### 4.6 Dual-mode systems

We are going to experiment with an additional syntax that I call a dual-mode proof system. (It is new to best of my knowledge, but I’d appreciate a literature check or feedback in that regard.) A dual-mode proof system $D\Pi$ specifies a proof generation algorithm $D\Pi.P$ and a proof verification algorithm $D\Pi.V$ that are just like those of a proof system in syntax. The difference is the CRS generator $D\Pi.C$, which now takes an input $\mu \in \{0, 1\}$ called the mode and returns a pair $\langle \text{crs}, \text{td} \rangle$ consisting of a common reference string and a trapdoor.

A dual-mode proof system $D\Pi$ gives rise to two (standard) proof systems that we call the proof systems induced by $D\Pi$ and denote $\Pi_1$ and $\Pi_0$. Their proof generation and verification algorithms are those of $D\Pi$, meaning $\Pi_{\mu}.P = D\Pi.P$ and $\Pi_{\mu}.V = D\Pi.V$, for both $\mu \in \{0, 1\}$. The difference between the two proof systems is in their CRS generation algorithms. Namely $\Pi_{\mu}.C$ is defined by: $\langle \text{crs}, \text{td} \rangle \leftarrow D\Pi.C(\mu)$; Return crs. The definitions we have above for proof systems now apply, or can be used for, either $\Pi_1$ or $\Pi_0$.

The value of dual-mode proof systems emerges when the common reference strings created in the two modes are indistinguishable. To formalize this, consider game $G_{\text{DND}}$ of Figure 8 associated to dual-mode proof system $D\Pi$, and let the mode advantage of adversary $A$ be defined by $\text{Adv}_{\text{DND}}(A) = 2 \Pr[G_{\text{DND}}^\text{mode}(A)] - 1$.

Theorem 4.6 says that if one of the induced proof systems $\Pi_\mu$ satisfies ZK with respect to some simulator $S$, and the modes are indistinguishable, then the other induced proof system $\Pi_{1-\mu}$ also satisfies ZK with respect to the same $S$. The statement is in the concrete security framework. Given

---

**Figure 7:** Games defining joint ZK + soundness/extractability of proof system $\Pi$.

<table>
<thead>
<tr>
<th>Game $G_{\text{DND}}^{\text{and}}_{\Pi,CV,S}$</th>
<th>Game $G_{\text{DND}}^{\text{ext}}_{\Pi,CV,S}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Init:</strong></td>
<td><strong>Init:</strong></td>
</tr>
<tr>
<td>1 $b \leftarrow {0,1}$</td>
<td>1 $b \leftarrow {0,1}$</td>
</tr>
<tr>
<td>2 If ($b = 1$) then $\text{crs} \leftarrow \Pi.C$</td>
<td>2 If ($b = 1$) then $\text{crs} \leftarrow \Pi.C$</td>
</tr>
<tr>
<td>3 Else ($\text{crs,td} \leftarrow S.C$</td>
<td>3 Else ($\text{crs,td} \leftarrow S.C$</td>
</tr>
<tr>
<td>4 Return $\text{crs}$</td>
<td>4 Return $\text{crs}$</td>
</tr>
<tr>
<td><strong>Pr</strong>($x,w$):</td>
<td><strong>Pr</strong>($x,w$):</td>
</tr>
<tr>
<td>5 If ($\text{CV(crs, x, w) = false}$) then return ⊥</td>
<td>5 If ($\text{CV(crs, x, w) = false}$) then return ⊥</td>
</tr>
<tr>
<td>6 If ($b = 1$) then $\text{pf} \leftarrow \Pi.P(\text{crs, x, w})$</td>
<td>6 If ($b = 1$) then $\text{pf} \leftarrow \Pi.P(\text{crs, x, w})$</td>
</tr>
<tr>
<td>7 Else $\text{pf} \leftarrow S.P(\text{crs, td, x})$</td>
<td>7 Else $\text{pf} \leftarrow S.P(\text{crs, td, x})$</td>
</tr>
<tr>
<td>8 Return $\text{pf}$</td>
<td>8 Return $\text{pf}$</td>
</tr>
<tr>
<td><strong>Vf</strong>($x,\text{pf}$):</td>
<td><strong>Ex</strong>($x,\text{pf}$):</td>
</tr>
<tr>
<td>9 If ($x \in \text{TrCl}_{\text{cv}}(\text{pars})$) then return false</td>
<td>10 If ($\langle x, \text{pf} \rangle \in Q$) then return false</td>
</tr>
<tr>
<td>10 If ($b = 1$) then return false</td>
<td>11 If ($\Pi.V(\text{crs, x, pf}) = \text{false}$) then return false</td>
</tr>
<tr>
<td>11 Return $\Pi.V(\text{crs, x, pf})$</td>
<td>12 If ($b = 1$) then Return true</td>
</tr>
<tr>
<td><strong>Fnt</strong>($b'$):</td>
<td>13 $w \leftarrow S.X(\text{crs, td, x, pf})$</td>
</tr>
<tr>
<td>12 Return ($b' = b$)</td>
<td>14 Return $\text{CV}(\text{crs, x, w})$</td>
</tr>
<tr>
<td></td>
<td><strong>Fnt</strong>($b'$):</td>
</tr>
<tr>
<td></td>
<td>15 Return ($b' = b$)</td>
</tr>
</tbody>
</table>
an adversary $A_{zk}$ against the ZK of $\Pi_{1-\mu}$, the reduction can be viewed as constructing two adversaries, one against the ZK of $\Pi_{\mu}$ and the other, called $A_{\text{mode}}$, against the mode indistinguishability of $D\Pi$, but the first construction is trivial, returning $A_{zk}$ itself.

**Theorem 4.6** Let $D\Pi$ be a dual-mode proof system for claim validator $CV$, and let $\Pi_{1}, \Pi_{0}$ be the induced proof systems. Let $\mu \in \{0, 1\}$. Given an adversary $A_{zk}$, we can construct an adversary $A_{\text{mode}}$ (shown explicitly in Figure 8), such that

$$\text{Adv}_{\Pi_{1-\mu}, CV, S}(A_{zk}) \leq \text{Adv}_{\Pi_{\mu}, CV, S}(A_{zk}) + \text{Adv}_{D\Pi}^{\text{mode}}(A_{\text{mode}}).$$

Adversary $A_{\text{mode}}$ has about the same running time as adversary $A_{zk}$.

**Exercise 4.7** Prove Theorem 4.6.

**Exercise 4.8** Develop results analogous to Theorem 4.6 for as many of the other notions as possible.

We will see later how the dual-mode perspective underlies proof system constructions in the literature and enables us to both better understand them and to state stronger results about them.

### 4.7 Relations

The following says that for SND1, security against adversaries making one $V_{F}$ query implies it against adversaries making $q$ queries, up to a factor $q$ increase in advantage. Asymptotically, this
means that SND1 can restrict attention to single-query adversaries. Concretely, in applications requiring security against many-query adversaries, we can factor in the degradation if starting from security results about single-query adversaries. More interestingly, it opens the door to tighter (better) reductions for particular proof systems that do not lose this factor of $q$, allowing efficiency gains through instantiation with smaller security parameters.

**Proposition 4.9** Let $\text{TC}$ be a true-claim language and $\Pi$ a proof system. Let $A$ be an adversary making $q$ queries to its $\text{VF}$ oracle. Then we can construct an adversary $A_1$ (shown explicitly in Figure 9), making only one query to its $\text{VF}$ oracle, such that

$$\text{Adv}^\text{snd}_{\Pi,\text{TC}}(A) \leq q \cdot \text{Adv}^\text{snd}_{\Pi,\text{TC}}(A_1).$$

The running time of $A_1$ is about that of $A$.

The proof would appear to be standard. Namely, let $A_1$ make a random guess $g \leftarrow [1..q]$, forward the $g$-th $\text{VF}$ query of $A$ to its own oracle, and answer the other queries on its own. This, however, runs into a difficulty, namely that at line 1 of Figure 4, oracle $\text{VF}$ tests membership of $x$ in $\text{TC}(\text{crs})$, and $A_1$ cannot do this test, since it may not be (and for proof systems of interest, isn’t) efficiently doable. (In asymptotic language, it is not polynomial time.) The proof below gets
around this before doing the guessing.

**Proof of Proposition 4.9:** Games $G_0, G_1$ of Figure 9 index the adversary’s oracle queries for future reference. Game $G_0$ includes the boxed code. As a consequence we have

$$
\text{Adv}^{\text{snd}}_{\Pi, \text{TC}}(A) = \Pr[G_0(A)].
$$

Games $G_0, G_1$ are identical-until-win. Lemma 3.1 thus implies that they set win with the same probability. However, win is also what they return, and once win is set to true it is never set back to false, so the games return true with the same probability, meaning

$$
\Pr[G_0(A)] = \Pr[G_1(A)].
$$

In game $G_1$, the line 5 reply to any $V_f$ query is false. This is what the game-playing has accomplished, namely it has allowed us to move to a game where we can reply to oracle queries in a trivial way and in particular without needing to test membership in TC.

Assume wlog that $A$ always makes exactly $q$ oracle queries. (Not fewer.) Games $G_{2, \ell}$ of Figure 9 are defined for all $\ell \in [1..q]$. The line 3 test in game $G_1$ is pushed, in these games, to FIN, with an additional twist, namely that it is performed on only one of the queries, namely the $\ell$-th. Then

$$
\text{Adv}^{\text{snd}}_{\Pi, \text{TC}}(A_1) = \frac{1}{q} \sum_{i=1}^{q} \Pr[G_{2,i}(A)]
$$

$$
\geq \frac{1}{q} \Pr[G_1(A)],
$$

where the last inequality is by the union bound.

**Proposition 4.10** Let $CV$ be a claim validator and $\Pi$ a proof system. Let $A$ be an adversary making $q$ queries to its Ex oracle. Then we can construct an adversary $A_1$ (shown explicitly in Figure 10), making only one query to its Ex oracle, such that

$$
\text{Adv}^{\text{ext1}}_{\Pi, CV}(A) \leq q \cdot \text{Adv}^{\text{ext1}}_{\Pi, CV}(A_1).
$$

The running time of $A_1$ is about that of $A$.

**Proof of Proposition 4.10:** Games $G_0, G_1$ of Figure 10 index the adversary’s oracle queries for future reference. Game $G_0$ includes the boxed code. As a consequence we have

$$
\text{Adv}^{\text{ext1}}_{\Pi, CV}(A) = \Pr[G_0(A)].
$$

Games $G_0, G_1$ are identical-until-win. Lemma 3.1 thus implies that they set win with the same probability. However, win is also what they return, and once win is set to true it is never set back to false, so the games return true with the same probability, meaning

$$
\Pr[G_0(A)] = \Pr[G_1(A)].
$$

In game $G_1$, the line 5 reply to any $V_f$ query is true. This is what the game-playing has accomplished, namely it has allowed us to move to a game where we can reply to oracle queries in a trivial way and in particular without needing to test whether a witness is valid, which an adversary cannot do since it does not have the witness.

Assume wlog that $A$ always makes exactly $q$ oracle queries. (Not fewer.) Games $G_{2, \ell}$ of Figure 10
Games $[G_0, G_1]$

**Init()**:  
1. $(\text{crs}, \text{td}) \leftarrow S.C$ ; Return crs

**Ex(x, pf)**:  
2. If $(\Pi.V(\text{crs}, x, pf) = \text{false})$ then return false
3. $i \leftarrow i + 1$ ; $x_i \leftarrow x$ ; $pf_i \leftarrow pf$ ; $\text{vf}_i \leftarrow \text{true}$
4. $w \leftarrow S.X(\text{crs}, \text{td}, x, pf)$
5. If $(\text{CV}(\text{crs}, x, w) = \text{false})$ then
6. $\text{win} \leftarrow \text{true}$ ; $\text{vf}_i \leftarrow \text{false}$
7. Return $\text{vf}_i$

**Fin()**:  
8. Return win

Game $G_{2,\ell}$

**Init()**:  
1. $\text{crs} \leftarrow \Pi.C$ ; Return crs

**Ex(x, pf)**:  
2. If $(\Pi.V(\text{crs}, x, pf) = \text{false})$ then return false
3. $i \leftarrow i + 1$ ; $x_i \leftarrow x$ ; $pf_i \leftarrow pf$
4. Return true

**Fin()**:  
5. $w \leftarrow S.X(\text{crs}, \text{td}, x_\ell, pf_\ell)$
6. Return $(\text{CV}(\text{crs}, x_\ell, w) = \text{false})$

<table>
<thead>
<tr>
<th>Adversary $A_1$:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $\text{crs} \leftarrow G_{\Pi, \text{CV}}^{\Pi, \text{Init}}$ ; $A_{\text{Init, Ex, Fin}}$</td>
<td></td>
</tr>
<tr>
<td><strong>Init()</strong>:</td>
<td></td>
</tr>
<tr>
<td>2. Return crs</td>
<td></td>
</tr>
</tbody>
</table>

**Ex(x, w)**:  
3. If $(\Pi.V(\text{crs}, x, pf) = \text{false})$ then return false
4. $i \leftarrow i + 1$ ; $x_i \leftarrow x$ ; $pf_i \leftarrow pf$
5. Return true

**Fin()**:  
6. $g \leftarrow [1..q]$  
7. $G_{\Pi, \text{CV}}^{\Pi, \text{Ex}}(x_g, pf_g)$  
8. $G_{\Pi, \text{CV}}^{\Pi, \text{Fin}}$

Figure 10: Games and adversary for Proposition 4.10.

are defined for all $\ell \in [1..q]$. The line 5 test in game $G_1$ is pushed, in these games, to Fin, with an additional twist, namely that it is performed on only one of the queries, namely the $\ell$-th. Then

$$\text{Adv}_{\Pi, \text{CV}}^{\Pi, \text{Init}}(A_1) = \frac{1}{q} \cdot \sum_{i=1}^{q} \Pr[G_{2,\ell}(A)]$$

$$\geq \frac{1}{q} \cdot \Pr[G_1(A)] ,$$

where the last inequality is by the union bound.

5 Groups and bilinear maps

If $G$ is a group, then its identity element is denoted $1_G$. If $h \in G$ then $(h)_G = \{ h^i : i \in \mathbb{Z} \}$ is the cyclic subgroup of $G$ generated by $h$. We let $\text{ord}_G(h) = |(h)_G| \geq 1$ be the order of $h$ in $G$, which
is also the smallest positive integer \( m \) such that \( h^m = 1_G \). Recall that \( \text{ord}_G(h) \) divides the order \( n = |G| \) of the group for all \( h \in G \). We say that \( g \in G \) is a generator of \( G \) if \( \langle g \rangle_G = G \), and denote by \( \text{Gen}(G) \) the set of all generators of \( G \). The group \( G \) is cyclic if \( \text{Gen}(G) \neq \emptyset \). If \( g \in \text{Gen}(G) \) and \( A \in G \) then \( \text{dlog}_{G,g}(A) \in \mathbb{Z}_n \) denotes the discrete logarithm of \( A \) to base \( g \), meaning \( A = g^{\text{dlog}_{G,g}(A)} \).

The following is sometimes called the fundamental theorem of cyclic groups.

**Proposition 5.1** Let \( G \) be any cyclic group of order \( n \). Let \( g \in \text{Gen}(G) \) and let \( m \) be any divisor of \( n \). Then:

1. Every subgroup of \( G \) is cyclic.
2. \( G \) has exactly one subgroup of order \( m \), and \( g^{n/m} \) generates it.

If \( m \) is a divisor of the order of cyclic group \( G \), then we will denote by \( G_m \) the unique (cyclic) subgroup of \( G \) of order \( m \).

**Exercise 5.2** Prove Proposition 5.1.

Let \( G, T \) be cyclic groups, both of the same order \( n \). We refer to \( T \) as the target group. The order \( n \) of the groups may be prime or composite. A bilinear map (also called a pairing) is a function \( \textbf{e}: \mathbb{G} \times \mathbb{G} \to T \) with the following properties:

1. **Bilinearity:** \( \textbf{e}(A, B_1 B_2) = \textbf{e}(A, B_1) \cdot \textbf{e}(A, B_2) \) for all \( A, B_1, B_2 \in \mathbb{G} \), and \( \textbf{e}(A_1 A_2, B) = \textbf{e}(A_1, B) \cdot \textbf{e}(A_2, B) \) for all \( A_1, A_2, B \in \mathbb{G} \).
2. **Non-degeneracy:** There exist \( A, B \in \mathbb{G} \) such that \( \textbf{e}(A, B) \neq 1_T \).

**Proposition 5.3** Let \( \textbf{e}: \mathbb{G} \times \mathbb{G} \to T \) be a bilinear map, and let \( n = |G| = |T| \). Then

1. \( \textbf{e}(g_1^{x_1}, g_2^{x_2}) = \textbf{e}(g_1, g_2)^{x_1 x_2} \) for all \( g_1, g_2 \in \mathbb{G} \) and all \( x_1, x_2 \in \mathbb{Z} \).
2. If \( g \in \text{Gen}(G) \) then \( \textbf{e}(g, g) \in \text{Gen}(T) \).
3. \( \textbf{e}(A, B) = \textbf{e}(B, A) \) for all \( A, B \in \mathbb{G} \).
4. \( \textbf{e}(A, 1_G) = \textbf{e}(1_G, A) = 1_T \) for all \( A \in \mathbb{G} \).
5. Let \( g \in \mathbb{G} \). Define \( f_g: \mathbb{G} \to T \) by \( f_g(A) = \textbf{e}(g, A) \). Then \( f_g \) is a group homomorphism, and if \( g \in \text{Gen}(G) \), it is an isomorphism. Likewise for \( f_g: \mathbb{G} \to T \) defined by \( f_g(A) = \textbf{e}(A, g) \).

**Exercise 5.4** Prove Proposition 5.3.

We will rely on the conjectured computational hardness of various problems about groups equipped with a bilinear map. The historically first of these problems was the Computational Bilinear Diffie Hellman (CBDH) problem [BF03]. The game picks \( g \leftarrow \text{Gen}(G) \) and \( x, y, z \leftarrow \mathbb{Z}_n \), where \( n = |G| \), and gives \( g, g^x, g^y, g^z \) to the adversary. To win, the adversary must return \( \textbf{e}(g, g)^{xyz} \). Its cbdh-advantage is the probability that it wins.

Of interest to us is a different problem, called the Subgroup Decision (SUBD) problem [BGN05]. Here, the order \( n = pq \) of the (cyclic) groups \( G, T \) is a composite number that (like an RSA modulus) is a product of distinct, odd primes \( p, q \). The game picks \( g \leftarrow \text{Gen}(G) \) and a challenge
bit $b \leftarrow \{0, 1\}$. If $b = 1$, it lets $h \leftarrow \text{Gen}(\mathbb{G}_q)$ be a generator of the unique order-$q$ subgroup of $\mathbb{G}$, and otherwise it lets $h \leftarrow \text{Gen}(\mathbb{G})$ be a generator of the full group $\mathbb{G}$. It gives $g, h$ to the adversary, asking the latter to provide a guess $b' \in \{0, 1\}$ of the value of $b$, and the adversary wins if $b' = b$. The subd-advantage of the adversary is $2w - 1$, where $w$ is the probability that it wins.

If the adversary is given $q$, it is easy for it to win the above game. (Why?) So in the formalization, the adversary is not given $q$, meaning not given the factorization of the group order $n$. This leads us to ask how exactly the problem is formalized.

In defining the CBDH problem, we regarded the groups $\mathbb{G}, \mathbb{T}$, their order $n$ and the bilinear map $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{T}$ as fixed. This works for many problems, but not all, and in particular not for SUBD, because if $n = pq$ is fixed it is not clear in what sense $q$ can be viewed as hidden or unknown to the adversary. Instead, we draw the groups from some distribution. Doing this is the responsibility of a group generator $\text{GG}$, and it parameterizes the problem, so that we have SUBD not by itself, but with respect to $\text{GG}$.

As an analogy, when we define the Discrete Logarithm (DL) problem in a concrete security setting, we can do so with respect to a fixed (cyclic) group and generator. But when we define the RSA problem, it is with respect to an RSA generator.

In the literature, we typically see the setup written something like $(n, q, \mathbb{G}, \mathbb{T}, e) \leftarrow \text{GG}$. We will also use such notation, but it is worth understanding all that it shoves under the rug. The notation seems to say that the generator returns the groups and bilinear map, but this of course is not possible, for these objects are too big to be efficiently specified. What $\text{GG}$ actually returns is some short description string $\eta$ which implicitly specifies the groups and map through algorithms, associated to $\text{GG}$, that allow some set of desired operations to be performed. There is an algorithm for testing membership in the first group; given $\eta$ and a description “$A$” of a group element, it returns $\text{true}$ if $A$ is in the group and $\text{false}$ otherwise. The group $\mathbb{G}$ is now implicitly defined as the set of all $A$ such that this algorithm returns $\text{true}$ given $\eta$ and “$A$.” There is an algorithm to perform the group operation on this group, taking $\eta$ and the descriptions “$A$”, “$B$” of two group elements to return the description of the group element $AB$. There are analogous algorithms for the second group $\mathbb{T}$. There are algorithms for computing inverses in the groups. There is an algorithm that defines $e$ by computing it. There may be algorithms for sampling random elements or generators of these groups. And so on.

If we were to proceed in strict formality, we would make these algorithms explicit and invoke them in our games, but this would be rather tedious and hard to follow. The compromise is an
informal setup step written \((n, q, G, e) \leftarrow \mathcal{S} G\), followed by usage of the groups with the usual mathematical operations.

Our convention on group generators is that \((n, q, G, T, e) \leftarrow \mathcal{S} G\) means the groups are cyclic of order \(n\), the map \(e : G \times G \rightarrow T\) is bilinear and \(q\) is a prime dividing \(n\). We may then distinguish a few special classes of group generators. We say that \(GG\) is a prime-order group generator if \(n = q\). We say it is a 2-prime group generator if \(q\) and \(n/q\) are distinct, odd primes. This allows us to capture most generators of interest in a common notation. Most definitions apply to all classes of generators, but the hardness of the problem may depend on the choice of generator.

With these conventions, the game formalizing the SUBD problem relative to group generator \(GG\) is on the left in Figure 11, and the subd-advantage of adversary \(A\) is defined by

\[
\text{Adv}_{\text{subd}}^{GG}(A) = 2\Pr[G_{\text{subd}}^{GG}(A)] - 1.
\]

In usage, \(GG\) will be a 2-prime group generator.

On the right in Figure 11 is a game formalizing the Decision Linear (DLIN) problem [BBS04] relative to group generator \(GG\). The dlin-advantage of adversary \(A\) is defined by

\[
\text{Adv}_{\text{dlin}}^{GG}(A) = 2\Pr[G_{\text{dlin}}^{GG}(A)] - 1.
\]

Here \(GG\) could be either a prime-order group generator or a 2-prime group generator.

6 NIZKs from SUBD

In this section we consider the Groth, Ostrovsky and Sahai (GOS) [GOS06] NIZK that is based on the assumed hardness of the SUBD problem. We cast it as a dual-mode proof system, something that is implicit but not explicit in [GOS06].

The system makes use of the BGN method for commitment or encryption [BGN05]. Suppose \(n = pq\) is a product of distinct, odd primes \(p, q\). Proposition 5.1 says that \(G\) has a unique subgroup of order \(q\) that we denote \(G_q\), and this subgroup is itself cyclic. For \(g, h \in G\), define \(E_{g,h} : \mathbb{Z} \times \mathbb{Z} \rightarrow G\) by \(E_{g,h}(m, r) = g^m h^r\). We think of this function as encrypting message \(m\) under coins \(r\) to produce ciphertext \(c \leftarrow E_{g,h}(m, r)\), or, alternatively, as committing to \(m\) with randomness \(r\) to produce commitment \(c\), in the style of Pedersen [Ped92]. For \(m \in \mathbb{Z}\) let \(E_{g,h}(m) = \{ E_{g,h}(m, r) : r \in \mathbb{Z} \}\) be the set of ciphertexts corresponding to message \(m\).

The encryption is not necessarily uniquely reversible, but becomes so, at least in principle, if we restrict the message space. Let \(M \subseteq \mathbb{Z}_p\), and suppose \(m_1, m_2 \in M\) are distinct. Then \(E_{g,h}(m_1) \cap E_{g,h}(m_2) = \emptyset\). (Why?) So in principle, given \(c \in E_{g,h}(m)\) for some \(m \in M\), one can uniquely recover \(m\), and we denote this \(m\) by \(E_{g,h}^{-1}(c)\). There is not necessarily a way to efficiently compute this decryption, but if the set \(M\) of messages is small, one can decrypt \(c\) by exhaustive search, as follows:

For all \(m \in M\) do

If \(\left( (g^{-m}c)^q = 1_G \right)\) then return \(m\).

Exercise 6.1 Assume \(M \subseteq \mathbb{Z}_p\) and \(m \in M\). Prove that on input \(c \in E_{g,h}(m)\) the above procedure returns \(m\).

In the GOS NIZK, we will encrypt messages \(m\) that are either 0 or 1. The system begins with a protocol to show that a given ciphertext \(c\) is an encryption of either 0 or 1, in zero-knowledge, meaning, in particular, without revealing whether the decryption is 0 or it is 1. We will call this the Zero-or-One system. We start with an algebraic characterization of the property of encrypting
Assume the prover and verifier know $c$. This Lemma does not appear explicitly in [GOS06]; we have extracted it from claims present in the text and worked out a detailed proof.

**Lemma 6.2** Let $n = pq$ where $p, q$ are distinct primes. Let $G, T$ be cyclic groups of order $n$, and let $e: G \times G \to T$ be a bilinear map. Let $g \in \text{Gen}(G)$ and let $h \in \text{Gen}(G_q)$. For $m \in \mathbb{Z}$ let $E_{g,h}(m) = \{g^m h^r : r \in \mathbb{Z}\}$. Let $c \in G$. Then the following are equivalent:

1. $c \in E_{g,h}(0) \cup E_{g,h}(1)$
2. $\{c, cg^{-1}\} \cap G_q \neq \emptyset$
3. $e(c, cg^{-1})^q = 1_T$.

**Proof of Lemma 6.2:** Suppose 1 holds. So $c = g^m h^r$ for some $m \in \{0,1\}$ and some $r \in \mathbb{Z}$. If $m = 0$ then $c^q = g^{mq} h^rq = g^{mq} = 1_G$, so $c \in G_q$. If $m = 1$ then $(cg^{-1})^q = g^{ma-qh^r} = g^{q(m-1)} = 1_G$, so $cg^{-1} \in G_q$. So 2 holds.

Now assume 2 holds. If, on the one hand, $c \in G_q$ then Proposition 5.3 implies that $e(c, cg^{-1})^q = e(c^q, cg^{-1})^q = 1_T$. If, on the other hand, $cg^{-1} \in G_q$ then $e(c, cg^{-1})^q = e(c, (cg^{-1})^q) = e(c, 1_G) = 1_T$. So 3 holds.

Now assume 3 holds. Since $g \in \text{Gen}(G)$, there is a $k \in [0..n-1]$ such that $c = g^k$. Then $1_T = e(c, cg^{-1})^q = e(g^k, g^{k-1})^q = e(g,g)^{k(k-1)q}$. Since $e(g,g) \in \text{Gen}(T)$, it must be that $n \mid k(k-1)q$, which means $p \mid k(q-1)$. Since $p$ is prime, either $p \mid k$ or $p \mid (k-1)$. If $p \mid k$ then $n \mid kq$ so $c^q = g^{kq} = 1_G$. If $p \mid (k-1)$ then $n \mid (k-1)q$ so $(cg^{-1})^q = g^{(k-1)q} = 1_G$. So 2 holds.

Now assume 2 holds. Suppose $c \in G_q$. Then $c = h^r$ for some $r \in [0..q-1]$, so $c = g^0 h^r$, putting $c$ in $E_{g,h}(0)$. Now suppose $cg^{-1} \in G_q$. Then $cg^{-1} = h^r$ for some $r \in [0..q-1]$, so $c = g^1 h^r$, putting $c$ in $E_{g,h}(1)$. So 1 holds.

Assume the prover and verifier know $c$. The prover wants to show that item 1 of Lemma 6.2 is true. Item 3 suggests that the verifier can simply check that $e(c, cg^{-1})^q = 1_T$. But this requires the verifier to know $q$, and is then not ZK, because given $q$ one can, via the above procedure, decrypt $c \in E_{g,h}(0) \cup E_{g,h}(1)$ to obtain the underlying message $m = E_{g,h}^{-1}(c)$. Instead, we consider the dual-model proof system $\text{DPR}_{\text{box}}$ (we call it the Zero-or-One system), and corresponding claim validator $\text{CV}_{\text{box}}$, that are described via Figure 13.

Theorem 6.3 says that, assuming SUBD, the modes are indistinguishable.
Theorem 6.3  Let $\mathbb{G}$ be a group generator. Let $D\Pi^0or1$ be the associated zero-or-one dual-mode proof system as per the left of Figure 13. Given an adversary $A_{mode}$, we can construct an adversary $A_{subd}$ (shown explicitly in the bottom right of Figure 13), such that

$$\text{Adv}^\text{mode}_{D\Pi^0or1}(A_{mode}) \leq \text{Adv}^\text{subd}_{\mathbb{G}}(A_{subd}).$$

Adversary $A_{subd}$ has about the same running time as adversary $A_{mode}$.

Theorem 6.4 makes a few claims about this system. Recall that a dual-mode proof system $D\Pi^0or1$ induces two, standard proof systems $\Pi_0, \Pi_1$. The theorem says that $\Pi_1$ has perfect soundness, and $\Pi_0$ has perfect zero knowledge, both relative to $CV^{0or1}$.

Claim validator $CV^{0or1 msg}$, shown on the right in Figure 13, defines the same language as $CV^{0or1}$, meaning $\text{TrCl}_{CV^{0or1}}(crs) = \text{TrCl}_{CV^{0or1 msg}}(crs)$ for all crs, the difference being that the witness for $CV^{0or1}$ is both message and randomness, while for $CV^{0or1 msg}$ it is only the message. This makes a difference with regard to the proof of knowledge property.

Theorem 6.4 says $\Pi_1$ has perfect extractability for $CV^{0or1 msg}$, showing it is a proof of knowledge of the message underlying a ciphertext, even if not of the underlying randomness. This shows the value of establishing the proof of knowledge for a claim validator different from the main one, something that does not seem to explicitly be in the literature, at least to best of my (limited) knowledge. In particular [GOS06] has no claim about the “Zero-or-One” system being a proof of knowledge, possibly because it isn’t for $CV^{0or1}$. (They do show the proof of knowledge property for their circuit-satisfiability system, which we will get to in a bit.)

Theorem 6.4  Let $\mathbb{G}$ be a group generator. Let $D\Pi^0or1$ be the associated zero-or-one dual-mode proof system, and $CV^{0or1}, CV^{0or1 msg}$ the associated claim validators, as per the left of Figure 13. Let $TC = \text{TrCl}_{CV^{0or1}} = \text{TrCl}_{CV^{0or1 msg}}$ be the true-claim language common to both claim validators. Let $\Pi_1, \Pi_0$ be the proof systems induced by $D\Pi^0or1$. Then:

1. $\Pi_1$ and $\Pi_0$ satisfy perfect completeness for $CV^{0or1}$
2. $\Pi_1$ satisfies perfect soundness for $TC$: $\text{Adv}^{\text{and}}_{\Pi_1,TC}(A) = 0$ for all adversaries $A$.
3. $\Pi_0$ satisfies perfect zero knowledge for $CV^{0or1}$ relative to the simulator $S^{0or1}$ shown on the right of Figure 13: $\text{Adv}^{2k}_{\Pi_0,CV^{0or1},S^{0or1}}(A) = 0$ for all adversaries $A$.
4. $\Pi_1$ satisfies perfect extractability for $CV^{0or1 msg}$ relative to the extractor $E^{0or1}$ shown on the right of Figure 13: $\text{Adv}^{\text{xtl}}_{\Pi_1,CV^{0or1 msg},E^{0or1}}(A) = 0$ for all adversaries $A$.

Proof of Theorem 6.4:  For 1, let $\mu \in \{0, 1\}$. Suppose $(\text{crs}, \text{td}) \in [D\Pi^0or1, C(\mu)]$ and parse the CRS into its constituents, $((n, G, T, e), g, h) \leftarrow \text{crs}$. Suppose $c, w$ satisfy $CV^{0or1}(\text{crs}, c, w) = \text{true}$, meaning $c = g^m h^w$ where $w \in \{0, 1\} \times \mathbb{Z}$ parses as $(m, r) \leftarrow w$. Let $s, t, u, pf_1, pf_2, pf_3$ be as at lines 10,11 of Figure 13. Then the condition at line 14 is not true, so this line does not return false. Now, making extensive use of Proposition 5.3, we have

$$e(pf_1, pf_2) = e(h^s, u^{st}) = e(h^r, u^{st}) = e(h^r, u) = e(h^r, g^{2m-1}h^r).$$

Now consider the two choices for $m$ separately. If $m = 0$ then $e(h^r, g^{2m-1}h^r) = e(h^r, g^{-1}h^r) = e(c, cg^{-1})$. Also if $m = 1$ then $e(h^r, g^{2m-1}h^r) = e(h^r, gh^r) = e(gh^r, h^r) = e(c, cg^{-1})$. So in either
<table>
<thead>
<tr>
<th>Claim validator $\text{CV}^{\text{bor}1}(\text{crs}, c, w)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $((n, G, T, e), g, h) \leftarrow \text{crs}$ ; $(m, r) \leftarrow w$</td>
</tr>
<tr>
<td>2. If $(m \not\in {0,1})$ then return false</td>
</tr>
<tr>
<td>3. Return $(c = g^m h^r)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CRS generator $\text{DP}^{\text{bor}1}.C(\mu)$: // $\mu \in {0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. $(n, q, G, T, e) \leftarrow \text{GG}$</td>
</tr>
<tr>
<td>5. If $(\mu = 1)$ then $g \leftarrow \text{Gen}(G)$ ; $h \leftarrow \text{Gen}(G_q)$ ; $td \leftarrow q$</td>
</tr>
<tr>
<td>6. Else $h \leftarrow \text{Gen}(G)$ ; $\gamma \leftarrow Z_n^*$ ; $g \leftarrow h^\gamma$ ; $td \leftarrow (q, \gamma)$</td>
</tr>
<tr>
<td>7. $\text{crs} \leftarrow (\text{crs}, G, T, e, g, h)$ ; Return (crs, td)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof generator $\text{DP}^{\text{bor}1}.P(\text{crs}, c, w)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. $((n, G, T, e), g, h) \leftarrow \text{crs}$</td>
</tr>
<tr>
<td>9. $(m, r) \leftarrow w$</td>
</tr>
<tr>
<td>10. $s \leftarrow Z_n^*$ ; $t \leftarrow s^{-1} \mod n$ ; $u \leftarrow g^2m^{-1}h^r$</td>
</tr>
<tr>
<td>11. $\text{pf}_1 \leftarrow h^s$ ; $\text{pf}_2 \leftarrow u^t$ ; $\text{pf}_3 \leftarrow g^s$</td>
</tr>
<tr>
<td>12. $\text{pf} \leftarrow (\text{pf}_1, \text{pf}_2, \text{pf}_3)$ ; Return pf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof verifier $\text{DP}^{\text{bor}1}.V(\text{crs}, c, pf)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>13. $((n, G, T, e), g, h) \leftarrow \text{crs}$</td>
</tr>
<tr>
<td>14. If $(c \not\in G$ or $pf \not\in G^3)$ then return false</td>
</tr>
<tr>
<td>15. $(\text{pf}_1, \text{pf}_2, \text{pf}_3) \leftarrow pf$</td>
</tr>
<tr>
<td>16. $\text{vf}_1 \leftarrow (e(c, cg^{-1}) = e(\text{pf}_1, \text{pf}_2))$</td>
</tr>
<tr>
<td>17. $\text{vf}_2 \leftarrow (e(\text{pf}_1, g) = e(h, \text{pf}_3))$</td>
</tr>
<tr>
<td>18. Return $(\text{vf}_1 \land \text{vf}_2)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sim CRS generator $S^{\text{bor}1}.C$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>19. $(\text{crs}, td) \leftarrow \text{DP}^{\text{bor}1}.C(0)$</td>
</tr>
<tr>
<td>20. Return (crs, td)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sim proof generator $S^{\text{bor}1}.P(\text{crs}, td, c)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>21. $((n, G, T, e), g, h) \leftarrow \text{crs}$ ; $(q, \gamma) \leftarrow td$</td>
</tr>
<tr>
<td>22. $p \leftarrow n/q$</td>
</tr>
<tr>
<td>23. If $(\exists c \in {0,1})$ then</td>
</tr>
<tr>
<td>24. $A \leftarrow c$ ; $B \leftarrow cg^{-1}$</td>
</tr>
<tr>
<td>25. Else $A \leftarrow cg^{-1}$ ; $B \leftarrow c$</td>
</tr>
<tr>
<td>26. $a \leftarrow Z_n^*$ ; $b \leftarrow a^{-1}$</td>
</tr>
<tr>
<td>27. $\text{pf}_1 \leftarrow A^a$ ; $\text{pf}_2 \leftarrow B^b$ ; $\text{pf}_3 \leftarrow \text{pf}_1^q$</td>
</tr>
<tr>
<td>28. $\text{pf} \leftarrow (\text{pf}_1, \text{pf}_2, \text{pf}_3)$</td>
</tr>
<tr>
<td>29. Return $\text{pf}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Claim validator $\text{CV}^{\text{bor}1-\text{msg}}(\text{crs}, c, m)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>30. $((n, G, T, e), g, h) \leftarrow \text{crs}$</td>
</tr>
<tr>
<td>31. If $(m \not\in {0,1})$ then return false</td>
</tr>
<tr>
<td>32. Return $(\exists r \in \mathbb{Z} : c = g^m h^r)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ext CRS generator $E^{\text{bor}1}.C$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>33. $(\text{crs}, td) \leftarrow \text{DP}^{\text{bor}1}.C(1)$</td>
</tr>
<tr>
<td>34. Return (crs, td)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ext witness generator $E^{\text{bor}1}.X(\text{crs}, td, c, pf)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>35. $((n, G, T, e), g, h) \leftarrow \text{crs}$ ; $q \leftarrow td$</td>
</tr>
<tr>
<td>36. If $(c^q = 1_G)$ then $m \leftarrow 0$ else $m \leftarrow 1$</td>
</tr>
<tr>
<td>37. Return $m$</td>
</tr>
</tbody>
</table>

![Figure 13](image-url)

Figure 13: Left: Zero-or-One claim validator $\text{CV}^{\text{bor}1}$ and dual-mode proof system $\text{DP}^{\text{bor}1}$ (three algorithms). The inputs to the proof generator must satisfy $c \in G$ and $w \in \{0,1\} \times \mathbb{Z}$. Right: Simulator $S$ (two algorithms), a second claim validator $\text{CV}^{\text{bor}1-\text{msg}}$, and extractor $S$ (two algorithms), for Theorem 6.4.

---

case we have $e(\text{pf}_1, \text{pf}_2) = e(c, cg^{-1})$, meaning $\text{vf}_1 = \text{true}$ at line 16. Also

$$e(h, \text{pf}_3) = e(h, g^s) = e(h^s, g) = e(\text{pf}_1, g),$$

so $\text{vf}_2 = \text{true}$ at line 17. So line 18 returns true.

For 2, let $\text{crs} \in [\Pi_1, C]$ be the response to A’s INIT query in game $G^{\text{snd}1}_{\Pi_1, C}$ of Figure 4, and parse it as $((n, G, T, e), g, h) \leftarrow \text{crs}$. Let $c, pf$ be a VF query of A. (The role of $x$ in the game is played here by $c$.) Assume $c \in G$ and $pf \in G^3$, for otherwise line 14 of Figure 13 returns false and we are done, and parse $pf$ as at line 15. Define $f_g : G \rightarrow T$ by $f_g(A) = e(A, g)$. Assume $\text{vf}_1$ at line 16 is true, for otherwise game $G^{\text{snd}1}_\Pi$ does not set $\text{win}$ to true. Then

$$f_g(\text{pf}_1^q) = f_g(\text{pf}_1)^q = e(h, \text{pf}_3)^q = e(h^q, \text{pf}_3) = 1_T.$$

Proposition 5.3 says that $f_g$ is a group isomorphism, so it must be that $\text{pf}_1^q = 1_G$. Now assume $\text{vf}_2$
at line 17 is true, for otherwise game $G_{\Pi_1,C^{\text{xor}1}}^{\text{send}}$ does not set $\text{win}$ to true. Then

$$e(c, cg^{-1})^g = e(pf_1, pf_2)^g = e(pf_1, pf_2; e(1_G, pf_2) = 1_T.$$ 

So condition 3 of Lemma 6.2 is met. The Lemma now says that condition 1 is met as well, which means $c \in \text{TrCl}_{C^{\text{xor}1}}(\text{crs})$. So game $G_{\Pi_1,C}^{\text{send}}$ will not set $\text{win}$ to true.

For 3, consider the games in Figure 14. Assume adversary $A$ makes no trivial queries, so that any $\text{Pf}(c, w)$ query satisfies $C^{\text{xor}1}(\text{crs}, c, w) = \text{true}$ — that is, $c = g^m h^r \in G$ where $w = (m, r) \in \{0, 1\} \times \mathbb{Z}$ — allowing us to eliminate line 5 of game $G_{\Pi_0,C^{\text{xor}1},S}^{2k}$. Algorithms $\Pi_0,C$ and $S^{\text{xor}1}.C$ create $\text{crs}$ the same way, via $\text{D}^{\text{xor}1}.C(0)$, so the games do as well, retaining the trapdoor information $q, \gamma$. In game $G_0$, oracle $\text{Pf}$ computes its replies using $\text{D}^{\text{xor}1}.P$, while in game $G_2$ it does so using $S^{\text{xor}1}.P$, so

$$\text{Adv}^{2k}_{\Pi_0,C^{\text{xor}1},S}(A) = \text{Pr}[G_0(A)] - \text{Pr}[G_2(A)]$$

$$= (\text{Pr}[G_0(A)] - \text{Pr}[G_1(A)]) + (\text{Pr}[G_1(A)] - \text{Pr}[G_2(A)]) .$$

We claim that in $G_1$, either $\gamma m + r \in \mathbb{Z}_n$ or $\gamma(m - 1) + r \in \mathbb{Z}_n$, meaning either $\text{gcd}(\gamma m + r, n) = 1$ or $\text{gcd}(\gamma(m - 1) + r, n) = 1$. (Justifying this is left as an exercise.) This means $v$ is always in $\mathbb{Z}_n^*$ after line 11. So line 12 is equivalent to line 6 with regard to the distribution of $s$. (Why is it important for this that $v \in \mathbb{Z}_n^*$?) Thus $\text{Pr}[G_0(A)] = \text{Pr}[G_1(A)]$. To complete the proof we show that $\text{Pr}[G_1(A)] = \text{Pr}[G_2(A)]$. This is done by showing that $(A^a, B^b, A^\gamma) = (h^a, u^r, g^s)$ in game $G_2$. We consider two cases. The first is that $c \in \text{Gen}(G)$, which happens iff $\gamma m + r \in \mathbb{Z}_n^*$. Since $g = h^\gamma$, we have $c = g^m h^r = h^{\gamma m + r} = h^v$. So

$$A^a = c^a = h^{av} = h^s .$$

We show that $B^b = u^r$ by showing that the discrete logarithms of these quantities in base $h$ are the same. In the following, arithmetic is modulo $n$, and we use the fact that $m(m - 1) = 0$, which
is true because $m \in \{0, 1\}$:

\[
d\log_{G,h}(B^b) = b \cdot d\log_{G,h}(cg^{-1}) = s^{-1}v \cdot d\log_{G,h}(h^\gamma(m-1)+r) = tv(\gamma(m-1) + r)
\]

\[
= (\gamma(m-1) + r)(\gamma m + rt) = (\gamma m + \gamma (m-1) + r)rt = (\gamma(2m-1) + r)rt
\]

\[
= rt \cdot d\log_{G,h}(h^\gamma(2^{m-1}+r)) = rt \cdot d\log_{G,h}(g^{2^{m-1}h^r}) = d\log_{G,h}(u^{rt}).
\]

Also

\[
A^a = c^a = h^av = g^s.
\]

The second case is that $c \not\in \text{Gen}(G)$, which happens iff $\gamma m + r \not\in \mathbb{Z}_n^*$. We have $cg^{-1} = g^{m-1}h^r = h^{\gamma(m-1)+r} = h^v$. So

\[
A^a = (cg^{-1})^a = h^av = h^s.
\]

The other checks are left as an exercise.  

Theorem 6.4 states properties of two proof systems, $\Pi_1$ and $\Pi_0$. In usage, of course, one must make a choice of which of the two proof systems to deploy. If we deploy $\Pi_1$ then we have perfect completeness and soundness and, by combining Theorems 6.4, 6.3 and 4.6, computational zero knowledge. A concrete-security statement for the zero knowledge can be obtained from the stated theorems. If we deploy $\Pi_0$, then we have perfect completeness and zero-knowledge, and computational soundness, in an analogous way.

Now we turn to the circuit-satisfiability proof system from [GOS06]. It reduces to the zero-or-one system via the following Lemma.

**Lemma 6.5** [GOS06] Let $b_0, b_1, b_2 \in \{0, 1\}$. Then

\[
b_2 = \text{NAND}(b_0, b_1) \iff b_0 + b_1 + 2b_2 - 2 \in \{0, 1\}.
\]

**Proof of Lemma 6.5:** The table below enumerates all possible values of $b_0, b_1, b_2$ and for each, computes both $\text{NAND}(b_0, b_1)$ and $b_0 + b_1 + 2b_2 - 2$.

<table>
<thead>
<tr>
<th>$b_0$</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>NAND($b_0, b_1$)</th>
<th>$b_0 + b_1 + 2b_2 - 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

The claim of the lemma can be verified from the table.  

Consider a circuit that is a single NAND gate. Its input wires are 1, 2 and its output wire is 3. The prover has an assignment $w \in \{0, 1\}^2$ to the inputs and wants to create a correct encrypted
Claim validator $C_{\text{val}}(c, w)$:
1. $((n, G, T, e), g, h) \rightarrow c$; $(m, r) \rightarrow w$
2. $w \leftarrow \text{Ev}(C, w)$
3. Return $(w|C \ln + C.Gt] = 1)$

CRS generator $D_{\text{gen}}(C, \mu)$: // $\mu \in \{0, 1\}$
1. $(n, q, G, T, e) \leftarrow \text{Gen}(G)$
2. If ($\mu = 1$) then $g \leftarrow \text{Gen}(G)$; $h \leftarrow \text{Gen}(G_q)$; td $\leftarrow q$
3. Else $h \leftarrow \text{Gen}(G)$; $\gamma \leftarrow Z_n^*$; $g \leftarrow h^\gamma$; td $\leftarrow (q, \gamma)$
4. $c$ $\leftarrow ((n, G, T, e), g, h)$; Return $(c, \text{rs}, \text{td})$

Proof generator $D_{\text{gen}}.P(c, w)$:
1. $((n, G, T, e), g, h) \leftarrow c$
2. $m \leftarrow \text{Ev}(C, w)$; $\ell \leftarrow C.Ln + C.Gt$
3. $c[\ell] \leftarrow g$; $r[\ell] \leftarrow 0$
4. For $i = 1, \ldots, \ell - 1$ do
5. $p[i] \leftarrow D_{\text{gen}}.P(c, w)$
6. $r[i] \leftarrow Z_n^*$; $c[i] \leftarrow g^{w[i]} h^{[r[i]}
7. For G = C.Ln $+\,1, \ldots, \ell$ do
8. $m \leftarrow \text{NAND}(w[1], g, h)$
9. $c \leftarrow c.[11.(G)] \cdot c.[12.(G)] \cdot c[g]^2 \cdot g^{-2}$
10. $r \leftarrow r[11.(G)] + r[12.(G)] + 2r[G]$
11. $p[\ell + G] \leftarrow D_{\text{gen}}.P(c, m, r)$
12. Return $(c, p)$

Proof verifier $D_{\text{ver}}.V(c, (c, p)):
1. $((n, G, T, e), g, h) \leftarrow c$
2. $\ell \leftarrow C.Ln + C.Gt$
3. $v[\ell] \leftarrow (c[\ell] = g)$
4. For $i = 1, \ldots, \ell - 1$ do
5. $v[i] \leftarrow D_{\text{ver}}.V(c, i, p[i])$
6. For G = C.Ln $+\,1, \ldots, \ell$ do
7. $c \leftarrow c.[11.(G)] \cdot c.[12.(G)] \cdot c[g]^2 \cdot g^{-2}$
8. $v[\ell + G] \leftarrow D_{\text{ver}}.V(c, p[G])$
9. Return $(\forall i : v[i] = \text{true})$

Sim CRS generator $S_{\text{gen}}.C:
1. $(c, \text{rs}, \text{td}) \leftarrow D_{\text{gen}}.C(0)$
2. Return $(c, \text{rs}, \text{td})$

Sim proof generator $S_{\text{gen}}.P(crs, td, C)$:
1. $((n, G, T, e), g, h) \leftarrow c$; $(q, \gamma) \leftarrow \text{td}$
2. $\ell \leftarrow C.Ln + C.Gt$
3. $w[\ell] \leftarrow 1$; $c[\ell] \leftarrow g$
4. For $i = 1, \ldots, \ell - 1$ do
5. $w[i] \leftarrow 0$; $r[i] \leftarrow \text{NAND}(w[i], r[i])$
6. $p[\ell + G] \leftarrow S_{\text{gen}}.P(crs, \text{td}, c[i])$
7. For G = C.Ln $+\,1, \ldots, \ell$ do
8. $c \leftarrow c.[11.(G)] \cdot c.[12.(G)] \cdot c[g]^2 \cdot g^{-2}$
9. $p[\ell + G] \leftarrow S_{\text{gen}}.P(crs, \text{td}, c)$
10. Return $(c, p)$

Ext CRS generator $E_{\text{gen}}.C:
1. $(c, \text{rs}, \text{td}) \leftarrow D_{\text{gen}}.C(1)$
2. Return $(c, \text{rs}, \text{td})$

Ext witness generator $E_{\text{gen}}.X(crs, \text{td}, C, (c, p)):
1. $((n, G, T, e), g, h) \leftarrow c$
2. For $i = 1, \ldots, \ell$ do
3. $w[i] \leftarrow E_{\text{gen}}.X(crs, \text{td}, c[i], p[i])$
4. Return $w$

---

Figure 15: Left: Circuit satisfiability claim validator $C_{\text{val}}$ and dual-mode proof system $D_{\text{gen}}$ (three algorithms). The inputs to the proof generator are a circuit $C$ and an assignment $w \in \{0, 1\}^{C.Ln}$ to its variables. Right: Simulator $S$ (two algorithms) and extractor $S$ (two algorithms), for Theorem 6.6.
constructed by running the prover of Figure 13. Now the prover will also prove that the bit underlying $c[3]$ is indeed the NAND of the bits underlying the other two ciphertexts. According to Lemma 6.5, it needs to show that
\[ x = w[1] + w[2] + 2w[3] - 2 \in \{0, 1\} \]
where $r \leftarrow r_1 + r_2 + 2r_3$. Now it uses the zero-or-one system to generate a proof $p[4]$ that $c$ encrypts either a 0 or a 1. The proof that it outputs is $(c, p)$.

Again we cast it as a dual-mode proof system. The full system, and the associated claim validator, are on the left in Figure 15.

**Theorem 6.6** Let $GG$ be a group generator. Let $D\Pi^{cs}$ be the circuit satisfiability dual-mode proof system, and $CV^{cs}$ the associated claim validator, as per the left of Figure 15. Let $TC = TrCl_{CV^{cs}}$ be the true-claim language associated to $CV^{cs}$. Let $\Pi_1, \Pi_0$ be the proof systems induced by $D\Pi^{cs}$. Then:

1. $\Pi_1$ and $\Pi_0$ satisfy perfect completeness for $CV^{cs}$
2. $\Pi_1$ satisfies perfect soundness for $TC$: $\text{Adv}_{\Pi_1, TC}^{\text{snd}}(A) = 0$ for all adversaries $A$.
3. $\Pi_0$ satisfies perfect zero knowledge for $CV^{cs}$ relative to the simulator $S^{cs}$ shown on the right of Figure 13: $\text{Adv}_{\Pi_0, CV^{cs}, S^{cs}}^{\text{zk}}(A) = 0$ for all adversaries $A$.
4. $\Pi_1$ satisfies perfect extractability for $CV^{cs}$ relative to the extractor $E^{cs}$ shown on the right of Figure 13: $\text{Adv}_{\Pi_1, CV^{cs}, E^{cs}}^{\text{xt}}(A) = 0$ for all adversaries $A$.

**7 Signatures from NIZKs**

We explore different ways to build a signature scheme from a NIZK proof system. Depending on the strength assumed from the NIZK, we have different constructions that make different assumptions beyond the NIZK.

**7.1 Signature definitions**

A (digital) signature scheme $DS$ specifies algorithms for key-generation, signing and verifying, as follows. Via $(sk, vk) \leftarrow DS.K$, the signer generates a secret signing key $sk$ and public verification key $vk$. Via $\sigma \leftarrow DS.S(sk, vk, m)$, the signer generates a signature of a message $m \in \{0, 1\}^*$. Via $vf \leftarrow DS.V(vk, m, \sigma)$, the verifier deterministically generates a boolean decision as to the validity of $\sigma$. Correctness requires that $DS.V(vk, m, \sigma) = \text{true}$ for all $\sigma \in [DS.S(sk, vk, m)]$, all $(sk, vk) \in [DS.K]$ and all $m \in \{0, 1\}^*$.

The security metrics for a signature scheme are unforgeability (UF) [GMR88] and strong unforgeability (SUF). The games are in Figure 16. We let $\text{Adv}_{DS}^{\text{uf}}(A) = \Pr[G_{DS}^{\text{uf}}(A)]$ and $\text{Adv}_{DS}^{\text{uf}}(A) = \Pr[G_{DS}^{\text{uf}}(A)]$ be the corresponding advantages of adversary $A$. The following says that SUF security implies UF security.

**Proposition 7.1** Let $DS$ be a signature scheme and $A$ an adversary. Then
\[ \text{Adv}_{DS}^{\text{uf}}(A) \leq \text{Adv}_{DS}^{\text{uf}}(A). \]
Figure 16: Games defining UF and SUF security of a signature scheme DS.

**Exercise 7.2** Prove Proposition 7.1.

**Exercise 7.3** Show by counter-example that UF does not imply SUF.

### 7.2 Construction from ZK+XT2

Let $F: \{0,1\}^{\mathcal{E},\mathcal{K}} \times \mathcal{F.D} \rightarrow \{0,1\}^*$ be a family of functions that we assume is one-way as defined in Section 3, and let CV be the claim validator of Figure 17. Let Π be a proof system that satisfies (completeness and) zkxt for CV. Let $DS_1$ be the signature scheme whose algorithms are shown in Figure 17. Theorem 7.4 says that $DS_1$ is SUF secure. This shows that if we have a NIZK that meets the joint ZKXT notion (which is equivalent to ZK+XT2) then it is easy to build a secure signature scheme.

The theorem statement, being concrete, does not have language of the form above with regard to $F, \Pi$ and what is assumed about them. The one-wayness assumption on $F$ emerges in the term involving the ow-advantage of $A_{ow}$ in Equation (6). The ZKXT is captured by assuming a simulator $S$, the zkxt-advantage of an adversary $A_{zkxt}$ relative to it, showing up again in Equation (6), being viewed as small.

This construction is from [DHLW10, BMT14]. The proofs there used ZK and XT2 separately. Our proof from ZKXT simplifies the prior ones.

In the Theorem statement, $T(A)$ refers to the running time of an algorithm $A$. 

**Theorem 7.4** Let $F: \{0,1\}^{\mathcal{E},\mathcal{K}} \times \mathcal{F.D} \rightarrow \{0,1\}^*$ be a family of functions and let CV be the claim validator of Figure 17. Let Π be a proof system and $S$ a simulator for it. Let $DS_1$ be the signature scheme whose algorithms are shown in Figure 17. Let $A_{ds}$ be an adversary making $q$ queries to its SIGN oracle. Then we can construct adversaries $A_{ow}$ and $A_{zkxt}$ (shown explicitly in Figure 17) such that

$$
\text{Adv}_{DS_1}^\text{suf}(A_{ds}) \leq \text{Adv}_{F}^\text{ow}(A_{ow}) + \text{Adv}_{\Pi,\text{CV},S}^\text{zkxt}(A_{zkxt}).
$$

The running time of $A_{ds}$ is that of $A_{ds}$ plus $O(q \cdot T(S,P) + T(S,C) + T(S,X))$. The running time of $A_{zkxt}$ is that of $A_{ds}$ plus $O(T(F))$.

How does the choice of simulator influence security? The running time of the constructed algorithms...
Proof of Theorem 7.4: Consider the games of Figure 18. At lines 2,4, game G allows such statements to be made in simple ways. Variables, and their lengths vary across the executions involved. It would be nice to find language for this. Finding language for this is hard because the inputs to the algorithms are random points for the precise running times. One may ask, why not give precise running time estimates in points to explicit, pseudocode constructions of the adversaries, which become the final reference views? The notation \( T(A) \) depends on the running time of the simulator algorithms. Thus, the more efficient the simulator, the more security we have shown for the signature scheme. This tells us that improving simulator runtime is valuable.

The notation \( T(A) \) used for various algorithms \( A \), above, fails to reflect that this time may depend also on the lengths of the inputs provided to the algorithms. For this reason, amongst others, the running-time estimates for the constructed adversaries given in the theorem statement should be viewed as rough indicators only. It is because of this deficiency in the statement that the theorem points to explicit, pseudocode constructions of the adversaries, which become the final reference points for the precise running times. One may ask, why not give precise running time estimates in the theorems? Finding language for this is hard because the inputs to the algorithms are random variables, and their lengths vary across the executions involved. It would be nice to find language that allows such statements to be made in simple ways.

**Proof of Theorem 7.4:** Consider the games of Figure 18. At lines 2,4, game \( G_1 \) switches to...
using the simulator. In Fin, it runs the extractor. We have

\[ \text{Adv}_F^{\text{diff}}(A_{ds}) = \Pr[G_0(A_{ds})] \]
\[ = \Pr[G_1(A_{ds})] + (\Pr[G_0(A_{ds})] - \Pr[G_1(A_{ds})]) . \]

We claim to have designed the Figure 17 adversaries so that

\[ \Pr[G_1(A_{ds})] \leq \text{Adv}_F^{\text{ow}}(A_{ow}) \]
\[ \Pr[G_0(A_{ds})] - \Pr[G_1(A_{ds})] \leq \text{Adv}_{\Pi,\mathit{CV},S}^{\text{zkt}}(A_{\text{zkt}}) . \]

Verifying the above two equations would conclude the proof, so let's do that.

Adversary \( A_{ow} \) answers \( A_{ds} \)'s queries as does game \( G_1 \). It calls \( S.X \) whenever game \( G_1 \) does. (Possibly more often because it omits the checks done by the game, but this can only increase its advantage.) This justifies Equation (7).

Let \( b \) denote the challenge bit chosen at random in game \( G_{\Pi,\mathit{CV},S}^{\text{zkt}} \), and let \( b' \) denote the query of \( A_{\text{zkt}} \) to \( G_{\Pi,\mathit{CV},S}^{\text{zkt}}.\) Fin. Then

\[ \text{Adv}_{\Pi,\mathit{CV},S}^{\text{zkt}}(A_{\text{zkt}}) = \Pr [ b' = 1 \mid b = 1 ] - \Pr [ b' = 1 \mid b = 0 ] , \]

where the probabilities are in the execution of game \( G_{\Pi,\mathit{CV},S}^{\text{zkt}} \) with adversary \( A_{\text{zkt}} \). Now we claim

\[ \Pr[G_0(A_{ds})] = \Pr [ b' = 1 \mid b = 1 ] \]
\[ \Pr[G_1(A_{ds})] = \Pr [ b' = 1 \mid b = 0 ] . \]

If \( b = 1 \) then ex = vf = true, which justifies the first. If \( b = 0 \) then ex equals CV(crs, (K, Y, m), X'), so is true iff \( F(K, X') = Y \), which justifies the second. Applying Lemma 3.2, this justifies Equation (8).
Claim validator $CV$($crs,(K,Y,m),X'$):
1. Return ($F(K,X') = Y$)

Key-generation algorithm $DS_2,K$:
2. $K \leftarrow \{0,1\}^{F,k}$
3. $sk \leftarrow F.D ; Y \leftarrow F(K,sk)$
4. $crs \leftarrow \Pi.C ; vk \leftarrow (crs,Y)$
5. Return ($sk,vk$)

Signing algorithm $DS_2,S$($vk,vk,m$):
6. ($K,Y$) $\leftarrow vk$
7. $\sigma \leftarrow \Pi.P(crs,(K,Y,m),sk)$
8. Return $\sigma$

Verifying algorithm $DS_2,V$($vk,m,\sigma$):
9. ($K,Y$) $\leftarrow vk$
10. Return $\Pi.V(crs,(K,Y,m),\sigma)$

Adversary $A_{ow}$:
1. ($crs,td$) $\leftarrow S.C ; (K,Y) \leftarrow G_f^\text{ow}.\text{INIT} ; vk \leftarrow (crs,Y)$
2. $A_{\text{int,sign,fin}}^{\text{key}}$

INIT:
3. Return $vk$

SIGN($m$):
4. $\sigma \leftarrow \Pi.P(crs,td,(K,Y,m)) ; \text{Return } \sigma$

FIN($m,\sigma$):
5. $X' \leftarrow S.X(crs,td,(K,Y,m),\sigma)$
6. $G_f^\text{ow}.\text{FIN}(X')$

Adversary $A_{zkxt}$:
1. $crs \leftarrow G_{\Pi,\text{CV,}S}.\text{INIT}$
2. $K \leftarrow \{0,1\}^{F,k} ; sk \leftarrow F.D ; Y \leftarrow F(K,sk) ; vk \leftarrow (crs,Y)$
3. $A_{\text{int,sign,fin}}^{\text{key}}$

INIT:
4. Return $vk$

SIGN($m$):
5. $\sigma \leftarrow G_{\Pi,\text{CV,}S}.\text{Pf}((K,Y,m),sk)$
6. $S \leftarrow S \cup \{(m,\sigma)\} ; \text{Return } \sigma$

FIN($m,\sigma$):
7. $vf \leftarrow \Pi.V(crs,(K,Y,m),\sigma)$
8. If ((vf = false) or ((m,\sigma) $\in$ $S$)) then $G_{\Pi,\text{CV,}S}.\text{FIN}(0)$
9. $ex \leftarrow G_{\Pi,\text{CV,}S}.\text{Ex}((K,Y,m),\sigma)$
10. If (ex) then $b' \leftarrow 1$ else $b' \leftarrow 0$
11. $G_{\Pi,\text{CV,}S}.\text{FIN}(b')$

Figure 19: Signature scheme $DS_2$.

7.3 Construction from ZK+XT1

References


36


