A Unified Security Perspective on Legally Fair Contract Signing Protocols

Diana Maimuţ[1][0000−0002−9541−5705] and George Teşeleanu[1,2][0000−0003−3953−2744]

[1] Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
{diana.maimut,tgeorge}@dcti.ro
[2] Department of Computer Science
"Al.I.Cuza" University of Iaşi 700506 Iaşi, Romania,
george.teseleanu@info.uaic.ro

Abstract. Inspired by Maurer's universal zero knowledge (UZK) abstract perspective and building on legally fair contract signing protocols without keystones, we propose and analyze the security of the first UZK class of co-signing protocols. We construct our main idea considering the stringent issue of scheme compatibility which characterizes communication systems. Typical examples are the cases of certificates in a public key infrastructure and the general issue of upgrading the version of a system. Thus, working in a general framework may reduce implementation errors and save application development and maintenance time.

Keywords: Public key, zero knowledge, digital signature, co-signature protocol, legal fairness, security proofs.

1 Introduction

The main issue addressed by zero knowledge proofs (ZKP) is represented by *identification schemes* (entity authentication). Thus, building on the most important goal that a ZKP can achieve one may find elegant solutions to various problems that arise in different areas: digital cash, auctioning, Internet of Things (IoT), password authentication and so on.

A typical zero knowledge protocol involves a prover $Peggy$ which possesses a piece of secret information x associated with her identity and a verifier $Victor$ whose job is to check that $Peggy$ really owns x. Two classical examples of such protocols (proposed for smartcards) are the Schnorr protocol [17] and the Guillou-Quisquater protocol [11]. Working in an abstract framework, Maurer shows in [12] that the previously mentioned protocols are actually instantiations of the same one.

Inspired by Maurer's generic perspective, we considered of great interest extending the unification paradigm to contract signing protocols. Therefore, we construct our main idea considering the stringent issue of scheme compatibility

which characterizes communication systems. Typical examples are the cases of certificates in a public key infrastructure and the general issue of upgrading the version of a system. Thus, working in a general framework may reduce implementation errors and save application development (and maintenance) time.

Various contract signing schemes which fall into three different design categories were proposed during the last decades: *gradual release* [8–10, 14], *optimistic* [2, 3, 13] and *concurrent* [4] or *legally fair* [6] models. A typical co-signing protocol involves two (mutually distrustful) signing partners, *Alice* and *Bob* wishing to compute a common function on their private inputs.

Compared to older paradigms like *gradual release* or optimistic models, concurrent signatures or legally fair protocols do not rely on trusted third parties and do not require too much interaction between co-signers. As such features seem much more attractive for users, we further consider legally fair co-signing protocols (rather than older solutions) in our paper.

To the best of our knowledge, in this work we present the first unified class of legally fair co-signing protocols without keystones and prove its security. Thus, we provide the reader with a common theoretical framework. To be more precise, we propose a class of UZK based co-signing protocols that maintains the valuable properties[3] of the scheme presented in [6].

As digital signature schemes represent the core of modern contract signing protocols, we preserve this perspective and prove the security of our main result building on the unified digital signature scheme we propose in Section 3.

Outline. We introduce notations, definitions, schemes and protocols used throughout the paper in Section 2. We present a signature scheme inspired by Maurer's UZK paradigm in Section 3 and prove its security in Appendix A. In Section 4 we present our main result, namely a UZK based co-signing protocol built on the legally fair contract signing protocol of [6]. We conclude and discuss related open problems in Section 5.

2 Preliminaries

Notations. Throughout the paper, the notation $|S|$ denotes the cardinal of a set S. The action of selecting a random element x from a sample space X is denoted by $x \xleftarrow{\$} X$, while $x \leftarrow y$ represents the assignment of value y to variable x. The probability of the event E to happen is denoted by $Pr[E]$. The subset $\{0, \ldots, s\} \in \mathbb{N}$ is denoted by $[0, s]$. Let $h : \{0,1\}^* \rightarrow \mathcal{C}$ be a hash function, where $\mathcal{C} \subset \mathbb{N}$.

2.1 Groups

Let $(\mathbb{G}, \star)$ and $(\mathbb{H}, \otimes)$ be two groups. We assume that the group operations $\star$ and $\otimes$ are efficiently computable. We further consider a more restrictive set of

[3] legal fairness without keystones, guaranteed output delivery

initial conditions compared to [12], in the sense that we also assume that $\mathbb{G}$ is commutative[4].

Definition 1 (Homomorphism). *Let $f : \mathbb{G} \to \mathbb{H}$ be a function (not necessarily one-to-one). We say that f is a homomorphism if $f(x \star y) = f(x) \otimes f(y)$.*

Throughout the rest of the paper we consider f to be a homomorphism as well as a one-way function[5]. To be consistent with [12], we denote the value $f(x)$ by $[x]$. Note that given $[x]$ and $[y]$ we can efficiently compute $[x \star y] = [x] \otimes [y]$, due to the fact that f is a homomorphism. We further denote the set of public parameters by $\mathsf{pp} = (\mathbb{G}, \mathbb{H}, f, h)$.

2.2 Zero-Knowledge Protocols

Let $Q : \{0,1\}^* \times \{0,1\}^* \to \{\texttt{true}, \texttt{false}\}$ be a predicate. Given a value y, Peggy will try to convince Victor that she knows a value x such that $Q(y, x) = \texttt{true}$. We further recall a definition from [5]. This definition captures the notion that being successful in a protocol (P, V) implies having the knowledge of a value x such that $Q(y, x) = \texttt{true}$.

Definition 2 (Proof of Knowledge Protocol). *An interactive protocol (P, V) is a proof of knowledge protocol for predicate Q if the following properties hold*

- *Completeness: V accepts the proof when P has as input an x with $Q(y, x) = \texttt{true}$;*
- *Soundness: there is an efficient program K (called knowledge extractor) such that for any $\hat{P}$ (possibly dishonest) with non-negligible probability of making V accept the proof, K can interact with $\hat{P}$ and output (with overwhelming probability) an x such that $Q(y, x) = \texttt{true}$.*

Definition 3 (2-Extractable). *Let Q be a predicate for a proof of knowledge. A 3-move protocol[6] with challenge space $\mathcal{C}$ is 2-extractable if from any two triplets (r, c, s) and (r, c', s'), with distinct $c, c' \in \mathcal{C}$ accepted by $Victor$, one can efficiently compute an x such that $Q(y, x) = \texttt{true}$.*

According to [12], UZK (Figure 1) is a zero-knowledge protocol if the conditions from Theorem 1 are satisfied. If the challenge space $\mathcal{C}$ is small, then one needs several 3-move rounds to make the soundness error negligible. We further assume that UZK satisfies the conditions stated in Theorem 1.

Theorem 1. *If values $\ell \in \mathbb{Z}$ and $u \in \mathbb{G}$ are known such that $\gcd(c_0 - c_1, \ell) = 1$ for all $c_0, c_1 \in \mathcal{C}$ with $c_0 \neq c_1$ and $[u] = y^\ell$, then the protocol described in Figure 1 is 2-extractable. Moreover, a protocol consisting of α rounds is a proof of knowledge if $1/|\mathcal{C}|^\alpha$ is negligible, and it is a zero-knowledge protocol if $|\mathcal{C}|$ is polynomially bounded.*

[4] The group $\mathbb{G}$ is considered as being generic in [12].
[5] meaning that it is infeasible to compute x from $f(x)$
[6] in which *Peggy* sends r, *Victor* sends c, *Peggy* sends s

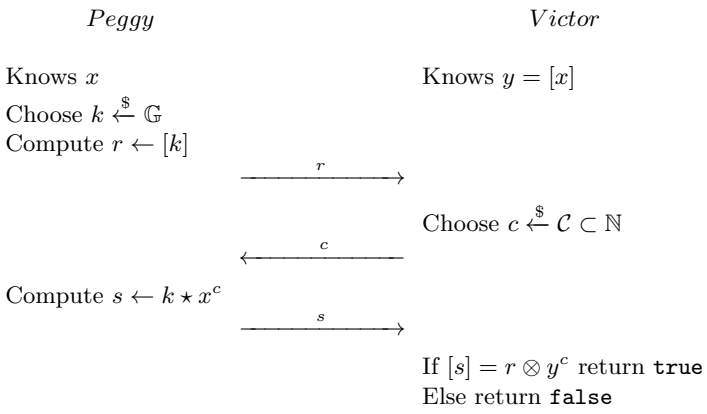

Fig. 1. Maurer's Unified Zero-Knowledge (UZK) Protocol.

2.3 Signatures

Definition 4 (Signature Scheme). *A Signature Scheme consists of three PPT algorithms:* KeyGen*,* Sign *and* Verify*. The first one takes as input a security parameter and outputs the system's parameters, the public key and the matching secret key. The secret key together with the* Sign *algorithm are used to generate a signature σ for a message m. Using the public key, the third algorithm verifies that a signature σ for a message m is generated using the matching secret key.*

Throughout the paper we only consider signature schemes which, on input m, produce triplets of the form $(\sigma_1, h(m\|\sigma_1), \sigma_2)$, independent of previous signatures. In these triplets we consider σ_2 as being dependent on m, σ_1 and $h(m\|\sigma_1)$. In some cases $h(m\|\sigma_1)$ is easily computable from the available data and, thus, can be omitted.

Lemma 1 (Forking Lemma). *Let $\mathcal{A}$ be a PPT algorithm, given only the public data as input. If A can find a valid signature $(m, \sigma_1, h(m\|\sigma_1), \sigma_2)$ with non-negligible probability, then, also with non-negligible probability, a replay of this machine with a different hashing oracle h' outputs two valid signatures $(m, \sigma_1, h(m\|\sigma_1), \sigma_2)$ and $(m, \sigma_1, h'(m\|\sigma_1), \sigma_2')$ such that $h(m\|\sigma_1) \neq h'(m\|\sigma_1)$.*

Security Model. We further present the security model of [16] for signature schemes.

Definition 5 (Signature Unforgeability - EF-CMA**).** *The notion of unforgeability for signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger:*

1. *The* KeyGen *algorithm is run and all the public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of signature queries to the challenger.*

3. Finally, $\mathcal{A}$ outputs a tuple $(m, \sigma_1, h(m\|\sigma_1), \sigma_2)$.

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, \sigma_1, h(m\|\sigma_1), \sigma_2) = \mathsf{True}$ *and $\mathcal{A}$ did not query the challenger on m. We say that a signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.*

2.4 Legally Fair Signatures without Keystones

In [6] the authors present a new contract signing paradigm that does not require keystones to achieve legal fairness. Their provably secure co-signature construction recalled in Figure 2 is based on Schnorr digital signatures [17].

In Figure 2, $\mathcal{L}$ represents a local non-volatile memory used by Bob and $\mathcal{C} = [0, q-1]$. During the protocol, Alice makes use of a publicly known auxiliary signature scheme σ_{x_A} using her secret key x_A.

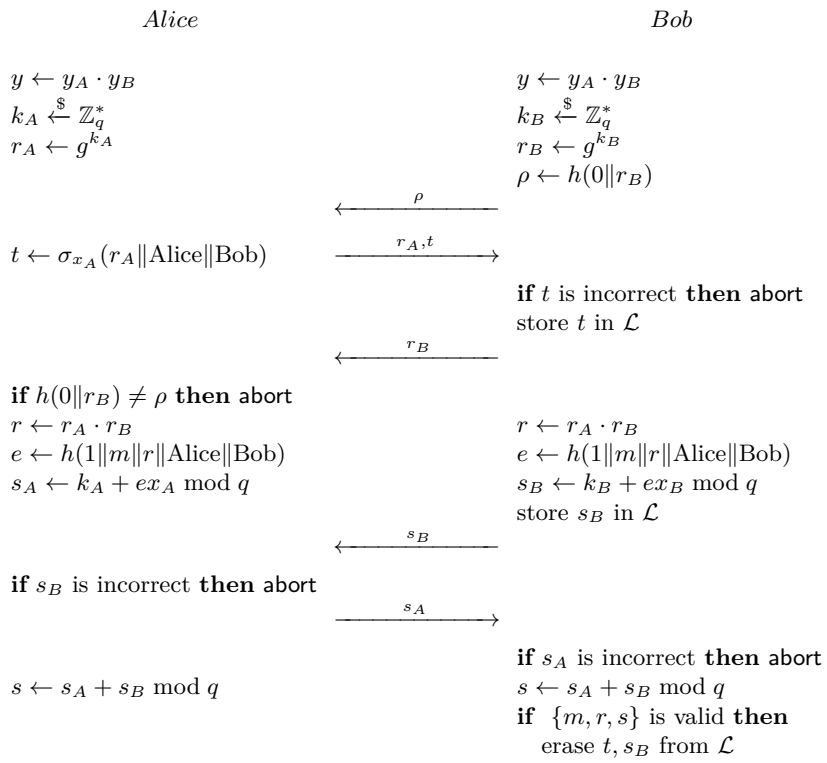

Fig. 2. The legally fair signature (without keystones) of message m.

Security model. According to the analysis presented in [6], a legally fair signature scheme is secure when it achieves existential unforgeability against an active

adversary $\mathcal{A}$ with access to an unlimited amount of conversations and valid co-signatures, *i.e.* $\mathcal{A}$ can perform the following queries:

- Hash queries: $\mathcal{A}$ can request the value of $h(x)$ for an x of his choosing.
- Sign queries: $\mathcal{A}$ can request a valid signature t for a message m and a public key y_C of his choosing.
- CoSign queries: $\mathcal{A}$ can request a valid co-signature (r, s) for a message m and a common public key $y_{C,D}$ of his choosing.
- Transcript queries: $\mathcal{A}$ can request a valid transcript $(m, \rho, r_C, t, r_D, s_C, s_D)$ of the co-signing protocol for a message m of his choosing, between users C and D of his choosing.
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.
- Directory queries: $\mathcal{A}$ can request the public key of any user.

The following definition captures the notion of unforgeability in the co-signing context:

Definition 6 (Co-Signature Unforgeability). *The notion of unforgeability for co-signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger:*

1. *The* KeyGen *algorithm is run and all the public parameters are provided to $\mathcal{A}$.*
2. *$\mathcal{A}$ can perform any number of queries to the challenger, as described above.*
3. *Finally, $\mathcal{A}$ outputs a tuple $(m, r, s, y_{C,D})$.*

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, r, s) = \mathsf{True}$ *and there exist public keys $y_C, y_D \in \mathcal{D}$ such that $y_{C,D} = y_C y_D$ and either of the following holds:*

- *$\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query CoSign on $(m, y_{C,D})$, and did not query Transcript on (m, y_C, y_D) nor (m, y_D, y_C).*
- *$\mathcal{A}$ did not query Transcript on (m, y_C, y_i) for any $y_i \neq y_C$ and did not query SKExtract on y_C, and did not query CoSign on (m, y_C, y_i) for any $y_i \neq y_C$.*

We say that a co-signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.

3 A UZK Based Digital Signature Scheme

We describe our proposed UZK based digital signature scheme (further referred to as UDS) in Table 1. We further provide the security margins of our scheme in Section 3.2.

3.1 Description

By applying the Fiat-Shamir transform [7] to the UZK protocol in Figure 1 we obtain the signature scheme presented in Table 1.

Table 1. A Unified Digital Signature (UDS) Scheme.

$\mathsf{KeyGen}(\mathsf{pp})$	On input the public parameters pp, this algorithm chooses uniformly at random $x \xleftarrow{\$} \mathbb{G}$ and computes $y \leftarrow [x]$. The output is the couple $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = x$ is kept private and $\mathsf{pk} = y$ is made public.
$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$	On input public parameters pp, a secret key sk, and a message m this algorithm selects a random $k \xleftarrow{\$} \mathbb{G}$, computes $r \leftarrow [k] \qquad e \leftarrow h(m\|r) \qquad s \leftarrow k \star x^e$ and outputs (r, s) as the signature of m.
$\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, (r, s))$	On input public parameters pp, a public key pk, a message m and a signature (r, s), this algorithm computes $e \leftarrow h(m\|r)$ and returns True iff $[s] = r \otimes y^e$; otherwise it returns False.

3.2 Security Analysis

The proofs presented in [15, 16] do not cover the generic case. Thus, we adapt the initial results to the UDS case and provide the reader with the proof of Theorem 2 in Appendix A.

Theorem 2. *If an* EF-CMA *attack on the UDS has non-negligible probability of success in the ROM, then the homomorphism* $[\cdot]$ *can be inverted in polynomial time.*

4 Main protocol

We describe our main result (a UZK class of legally fair contract signing protocols) in Figure 3 and discuss its correctness. We further prove the security of our proposed idea in Section 4.2 based on the security of the UDS scheme presented in Section 3.

Compared to the initial work on legally fair contract signing protocols without keystones [6], we give a more complete proof by taking into account the signature scheme σ too.

4.1 Description

To illustrate our unified paradigm, we now discuss a legally fair co-signing protocol built from the UDS (Figure 3), which produces signatures compatible with standard UDS (Table 1). This contract signing protocol is provably secure in the ROM assuming the one-way property of $[\cdot]$.

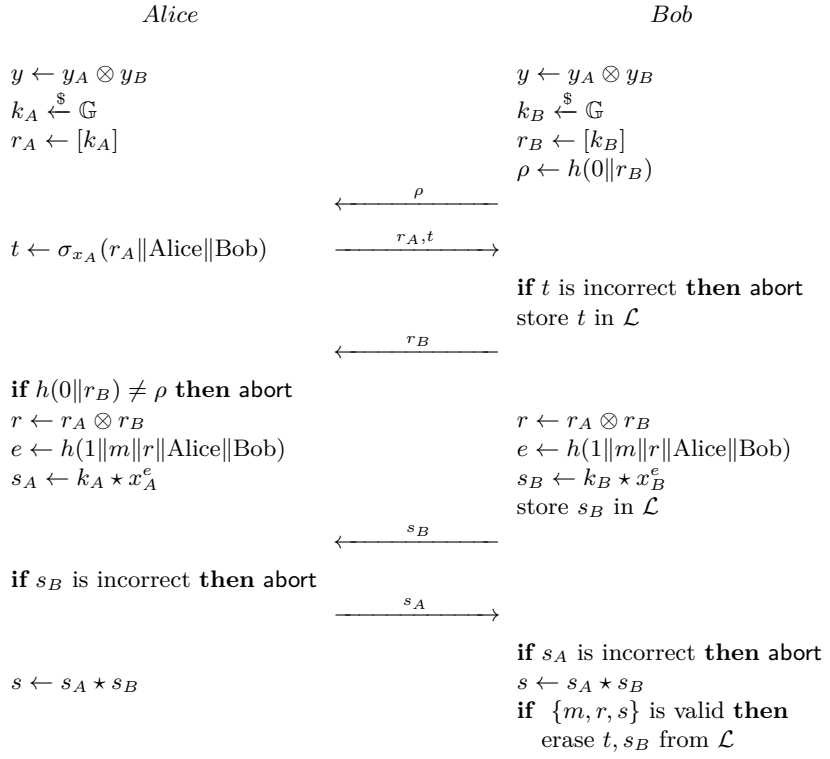

Fig. 3. A Class of Legally Fair Co-Signature Schemes.

Correctness. To prove the correctness of the co-signing schemes class described in Figure 3 we use the commutative property of $\mathbb{G}$ which is preserved by $f(x)$:

$$\begin{aligned}
[s] &= [s_A \star s_B] \\
&= [s_A] \otimes [s_B] \\
&= [k_A] \otimes [x_A]^e \otimes [k_B] \otimes [x_B]^e \\
&= [k_A] \otimes [k_B] \otimes ([x_A] \otimes [x_B])^e \\
&= r \otimes y^e.
\end{aligned}$$

4.2 Security Analysis

To prove that the unified co-signature protocol is secure in the ROM we use the following strategy: assuming $\mathcal{A}$ is an efficient forger for the co-signature scheme, we turn $\mathcal{A}$ into an efficient forger for UDS, then invoke Lemma 1 to prove the existence of an efficient inverter for the homomorphism $[\cdot]$. We further address two scenarios: when the attacker plays Alice's role, and when the attacker plays Bob's.

4.2.1 Adversary Attacks Bob

Theorem 3. *If $\mathcal{A}_{Alice}$ plays the role of Alice and is able to forge a co-signature with non-negligible probability, then we can construct an* EF-CMA *attack on the UDS that has non-negligible probability of success.*

Proof. The proof consists in constructing a simulator $\mathcal{S}_{\text{Bob}}$ that interacts with the adversary and forces it to actually produce a UDS forgery. Here is how this simulator behaves at each step of the protocol.

Key Establishment Phase. $\mathcal{S}_{\text{Bob}}$ is given a target public key y. As a simulator, $\mathcal{S}_{\text{Bob}}$ emulates not only Bob, but also all oracles and the directory $\mathcal{D}$ (see Figure 4).

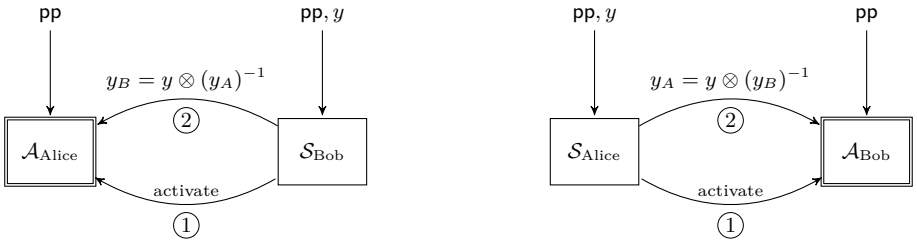

Fig. 4. The simulator $\mathcal{S}_{\text{Bob}}$ (left) or $\mathcal{S}_{\text{Alice}}$ (right) answers the attacker's queries to the public directory $\mathcal{D}$.

To inject a target $y \leftarrow [x]$ into $\mathcal{A}$, the simulator $\mathcal{S}_{\text{Bob}}$ reads $y_{\mathcal{A}}$ from $\mathcal{D}$ and poses as an entity whose public-key is $y_{\mathcal{S}_{\text{Bob}}} \leftarrow y \otimes (y_{\mathcal{A}})^{-1}$. It follows that $y_{\mathcal{A},\mathcal{S}_{\text{Bob}}}$, the common public-key of $\mathcal{A}$ and $\mathcal{S}_{\text{Bob}}$ will be precisely $y_{\mathcal{A},\mathcal{S}_{\text{Bob}}} \leftarrow y_{\mathcal{S}_{\text{Bob}}} \otimes y_{\mathcal{A}}$ which, by construction, is exactly y.

Then $\mathcal{S}_{\text{Bob}}$ activates $\mathcal{A}_{\text{Alice}}$, who queries the directory and gets y_B. At this point in time, $\mathcal{A}_{\text{Alice}}$ is tricked into believing that she has successfully established a co-signature public-key set (pp, y) with the "co-signer" $\mathcal{S}_{\text{Bob}}$.

Algorithm 1: Hashing oracle $\mathcal{O}_h$ simulation for h.

Input: A hashing query q_i from $\mathcal{A}$
if $\exists h_i, \{q_i, h_i\} \in T$ **then**
  $e \leftarrow h_i$
else
  $e \xleftarrow{\$} \mathcal{C}$
  Append $\{q_i, e\}$ to T
end if
return e

Query Phase. $\mathcal{A}_{\text{Alice}}$ will start to present queries to $\mathcal{S}$. Thus, $\mathcal{S}$ must respond to three types of queries: hash queries, co-signature queries and transcript queries.

$\mathcal{S}$ will maintain a table T containing all the hash queries performed throughout the attack. At start $T \leftarrow \varnothing$. We further describe the simulations of the hash function in Algorithm 1 and the co-signature protocol in Algorithm 2. When $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending $(m, \rho, r_A, t, r_B, s_B, s_A)$ from a previously successful interaction.

Output Phase. After performing queries, $\mathcal{A}_{\text{Alice}}$ eventually outputs a co-signature (r, s) valid for $y_{\mathcal{A},\mathcal{S}_{\text{Bob}}}$ where $r = r_A \otimes r_B$ and $s = s_A \star s_B$. By design, these parameters are those of a UDS and therefore $\mathcal{A}_{\text{Alice}}$ has produced a UDS forgery.

Algorithm 2: Co-signing oracle simulation for $\mathcal{S}_{\text{Bob}}$.

Input: A co-signature query m from $\mathcal{A}_{\text{Alice}}$

$s_B \xleftarrow{\$} \mathbb{G}$
$e \xleftarrow{\$} \mathcal{C}$
$r_B \leftarrow [s_B] \otimes y^{-e}$
Send $h(0\|r_B)$ to $\mathcal{A}_{\text{Alice}}$
Receive r_A, t from $\mathcal{A}_{\text{Alice}}$
Send r_B to $\mathcal{A}_{\text{Alice}}$
$r \leftarrow r_A \otimes r_B$
$u \leftarrow 1\|m\|r\|\text{Alice}\|\text{Bob}$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
  abort
else
  Append $\{u, e\}$ to T
end if
return s_B

To understand $\mathcal{S}_{\text{Bob}}$'s co-signature reply (Algorithm 2), assume that $\mathcal{A}_{\text{Alice}}$ is an honest Alice who plays by the protocol's rules. For such an Alice, (r, s) is a valid signature with respect to the co-signature public-key set $\{\mathsf{pp}, y\}$.

There is a case in which $\mathcal{S}_{\text{Bob}}$ aborts the protocol before completion: this happens when it turns out that $1\|m\|r\|\text{Alice}\|\text{Bob}\|t$ has been previously queried by $\mathcal{A}_{\text{Alice}}$. In that case, it is no longer possible for $\mathcal{S}_{\text{Bob}}$ to reprogram the oracle, which is why $\mathcal{S}_{\text{Bob}}$ must abort. Since $\mathcal{A}_{\text{Alice}}$ does not know the random value r_B, such a bad event would only occur with a negligible probability exactly equal to $q_h/|\mathcal{C}|$, where q_h is the number of queries to $\mathcal{O}_h$.

Therefore, $\mathcal{A}$ is turned into a forger for the SFS with probability $1 - q_h/|\mathcal{C}|$. As $\mathcal{A}$ has a success probability ϵ_{succ}, the success probability of $\mathcal{A}$ in the simulated environment is $\epsilon_{sim} = (1 - q_h/|\mathcal{C}|)\epsilon_{succ}$.

□

Corollary 1. *If $\mathcal{A}_{Alice}$ plays the role of Alice and is able to forge a co-signature with non-negligible probability, then the homomorphism $[\cdot]$ can be inverted in polynomial time.*

4.2.2 Adversary Attacks Alice

Theorem 4. *If $\mathcal{A}_{Bob}$ plays the role of Bob and is able to forge a co-signature with non-negligible probability, then we can construct an* EF-CMA *attack on the UDS that has non-negligible probability of success if signature σ_{x_A} can be simulated without knowing the secret key x_A.*

Proof. Here also the proof consists in constructing a simulator, $\mathcal{S}_{\text{Alice}}$, that interacts with the adversary and forces it to actually produce a UDS forgery. The simulator's behavior at different stages of the security game is as follows.

The Key Establishment Phase. $\mathcal{S}_{\text{Alice}}$ is given a target public key y. Again, $\mathcal{S}_{\text{Alice}}$ impersonates not only Alice, but also all the oracles and $\mathcal{D}$.

$\mathcal{S}_{\text{Alice}}$ injects the target y into the game as described in Theorem 3. Now $\mathcal{S}_{\text{Alice}}$ activates $\mathcal{A}_{\text{Bob}}$, who queries $\mathcal{D}$ (actually controlled by $\mathcal{S}_{\text{Alice}}$) to get y_A. $\mathcal{A}_{\text{Bob}}$ is thus tricked into believing that it has successfully established a co-signature public-key set (pp, y) with the "co-signer" $\mathcal{S}_{\text{Alice}}$.

Query Phase. $\mathcal{A}$ will start to present queries to $\mathcal{S}$. Thus, $\mathcal{S}$ must respond to four types of queries: hash queries, signature queries, co-signature queries and transcript queries. We consider oracles $\mathcal{O}_h$ as in Theorem 3. We denote by $\mathcal{O}_\sigma$ the simulation of σ_{x_A}. We further describe the simulation of the co-signature algorithm in Algorithm 3. When $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending $(m, \rho, r_A, t, r_B, s_B, s_A)$ from a previously successful interaction.

Output Phase. After performing queries, $\mathcal{A}_{\text{Bob}}$ eventually outputs a co-signature (r, s) valid for $y_{\mathcal{S}_{\text{Alice}}, \mathcal{A}_{\text{Bob}}}$ where $r = r_A \otimes r_B$ and $s = s_A \star s_B$. By design, these parameters are those of a UDS and therefore $\mathcal{A}_{\text{Bob}}$ has produced a UDS forgery.

As in Theorem 3, Algorithm 3 may fail with probability $q_h/|\mathcal{C}|$. Thus, the success probability of $\mathcal{A}$ in the simulated environment is $\epsilon_{sim} = (1 - q_h/|\mathcal{C}|)\epsilon_{succ}$. □

Corollary 2. *If $\mathcal{A}_{Bob}$ plays the role of Bob and is able to forge a co-signature with non-negligible probability, then the homomorphism $[\cdot]$ can be inverted in polynomial time if signature σ_{x_A} can be simulated without knowing the secret key x_A.*

5 Conclusion

In this paper we presented a signature scheme inspired by Maurer's UZK paradigm from [12] and proved its security. We further described our main result, *i.e.* a UZK class of co-signing protocols based on both Maurer's abstract perspective and the legally fair framework from [6] and then proved its security.

Open Problems. A couple of interesting related studies could be the analysis of our co-signature protocols' resistance to SETUP (Secretly Embedded Trapdoor with Universal Protection) attacks and the proposal of suitable countermeasures.

Algorithm 3: Co-signing oracle simulation for $\mathcal{S}_{\text{Alice}}$.

Input: A co-signature query m from $\mathcal{A}_{\text{Bob}}$
Receive ρ from $\mathcal{A}_{\text{Bob}}$
Query T to retrieve r_B such that $h(0\|r_B) = \rho$
$s_A \xleftarrow{\$} \mathbb{G}$
$e \xleftarrow{\$} \mathcal{C}$
$r \leftarrow r_B \otimes [s_A] \otimes y^{-e}$
$u_1 \leftarrow 1\|m\|r$
if $\exists e' \neq e, \{u_1, e'\} \in T$ **then**
  abort
else
  Append $\{u_1, e\}$ to T
end if
$r_A \leftarrow r \otimes r_B^{-1}$
$u_2 \leftarrow r_A\|\text{Alice}\|\text{Bob}$
$t \leftarrow \mathcal{O}_\sigma(u_2)$
Send r_A, t to $\mathcal{A}_{\text{Bob}}$
Receive r_B from $\mathcal{A}_{\text{Bob}}$
Receive s_B from $\mathcal{A}_{\text{Bob}}$
return s_A

References

1. Zero-Knowledge Proof: More secure than passwords? https://blog.ingenico.com/posts/2017/07/zero-knowledge-proof-more-secure-than-passwords.html
2. Asokan, N., Schunter, M., Waidner, M.: Optimistic Protocols for Fair Exchange. In: Proceedings of the 4^{th} ACM Conference on Computer and Communications Security - CCS'97. pp. 7–17. ACM (1997)
3. Cachin, C., Camenisch, J.: Optimistic Fair Secure Computation. In: Advances in Cryptology - CRYPTO'00. Lecture Notes in Computer Science, vol. 1880, pp. 93–111. Springer (2000)
4. Chen, L., Kudla, C., Paterson, K.G.: Concurrent Signatures. In: Advances in Cryptology - EUROCRYPT'04. Lecture Notes in Computer Science, vol. 3027, pp. 287–305. Springer (2004)
5. Feige, U., Fiat, A., Shamir, A.: Zero-Knowledge Proofs of Identity. Journal of cryptology **1**(2), 77–94 (1988)
6. Ferradi, H., Géraud, R., Maimuţ, D., Naccache, D., Pointcheval, D.: Legally Fair Contract Signing Without Keystones. In: International Conference on Applied Cryptography and Network Security - ACNS'16. Lecture Notes in Computer Science, vol. 9696, pp. 175–190. Springer (2016)
7. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: CRYPTO 1986. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
8. Garay, J., MacKenzie, P., Prabhakaran, M., Yang, K.: Resource Fairness and Composability of Cryptographic Protocols. In: Proceedings of the 3^{rd} Theory of Cryptography Conference - TCC'06. Lecture Notes in Computer Science, vol. 3876, pp. 404–428. Springer (2006)

9. Goldwasser, S., Levin, L., Vanstone, S.A.: Fair Computation of General Functions in Presence of Immoral Majority. In: Advances in Cryptology - CRYPT0'90. Lecture Notes in Computer Science, vol. 537, pp. 77–93. Springer (1991)
10. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete Fairness in Secure Two-Party Computation. Jornal of the ACM **58**(6), 1–37 (December 2011)
11. Guillou, L.C., Quisquater, J.J.: A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In: EUROCRYP 1988. Lecture Notes in Computer Science, vol. 330, pp. 123–128. Springer (1988)
12. Maurer, U.: Unifying Zero-Knowledge Proofs of Knowledge. In: AFRICACRYPT 2009. Lecture Notes in Computer Science, vol. 5580, pp. 272–286. Springer (2009)
13. Micali, S.: Simple and Fast Optimistic Protocols for Fair Electronic Exchange. In: Proceedings of the 22^{nd} Annual Symposium on Principles of Distributed Computing - PODC'03. pp. 12–19. ACM (2003)
14. Pinkas, B.: Fair Secure Two-Party Computation. In: Advances in Cryptology - EUROCRYPT'03. Lecture Notes in Computer Science, vol. 2656, pp. 87–105. Springer (2003)
15. Pointcheval, D., Stern, J.: Security Proofs for Signature Schemes. In: Advances in Cryptology - EUROCRYPT'96. Lecture Notes in Computer Science, vol. 1070, pp. 387–398. Springer (1996)
16. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology **13**(3), 361–396 (2000)
17. Schnorr, C.P.: Efficient Identification and Signatures For Smart Cards. In: Advances in Cryptology - CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer (1989)

A Proof of Theorem 2

If an attacker $\mathcal{A}$ can forge a UDS, then we are able to construct a simulator $\mathcal{S}$ that interacts with $\mathcal{A}$ and forces it to produce a forgery. By using Lemma 1 we transform $\mathcal{A}$ into a homomorphism inverter (*i.e.* that computes an x' such that $y = [x']$). We further show how $\mathcal{S}$ can simulate the three phases necessary to mount the EF-CMA attack.

Key Establishment Phase. In this phase $\mathcal{S}$ sets up the public key as $y = [x]$ and then activates $\mathcal{A}$ with input y.

Query Phase. $\mathcal{A}$ will start to present queries to the $\mathcal{S}$. Thus, $\mathcal{S}$ must respond to two types of queries: hash and signature queries. We consider oracle $\mathcal{O}_h$ as in Theorem 3. We further describe the simulations of the the signature scheme in Algorithm 4.

Output Phase. After the query phase, $\mathcal{A}$ will eventually produce a forgery (r, s).

When simulating the signing oracle $\mathcal{O}_S$ there is a case when $\mathcal{S}$ aborts before completion: this happens when $m\|r$ has already been queried by $\mathcal{A}$. In this case, $\mathcal{S}$ can not reprogram $\mathcal{O}_h$, which is why it must abort. Since $\mathcal{A}$ does not know the random value r, the previously described event occurs with a negligible probability q_h/q, where q_h is the number of queries to $\mathcal{O}_h$.

Therefore, $\mathcal{A}$ is turned into a forger for the UDS with probability $1 - q_h/q$. As $\mathcal{A}$ has a success probability ϵ_{succ}, the success probability of $\mathcal{A}$ in the simulated environment is $\epsilon_{sim} = (1 - q_h/q)\epsilon_{succ}$.

Let γ be the position of $m\|r$ in T from $\mathcal{O}_h$. After $\mathcal{A}$ produces a forgery (r, s), $\mathcal{S}$ runs $\mathcal{A}$ with the same inputs and a different h oracle (Algorithm 5). As before, $\mathcal{S}$ will maintain a table $\tilde{T}$ containing all the h queries performed throughout this phase of the attack. At start $\tilde{T} \leftarrow \varnothing$. Then, by Lemma 1, $\mathcal{A}$ will produce a different forgery (r, s'). Thus, we obtain $c = \mathcal{O}_h(m\|r) \neq \mathcal{O}'_h(m\|r) = c'$. Using the 2-extractable property of UZK, we obtain an x' such that $y = [x']$.

Algorithm 4: Signing oracle $\mathcal{O}_S$ simulation.

Input: A signature query m

$s \xleftarrow{\$} \mathbb{G}$
$e \xleftarrow{\$} \mathcal{C}$
$r \leftarrow [s] \otimes y^{-e}$
$u \leftarrow m\|r$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
  abort
else
  Append $\{u, e\}$ to T
end if
return (r, s)

Algorithm 5: Hashing oracle $\mathcal{O}'_h$ simulation for h.

Input: A hashing query q_i from $\mathcal{A}$, an index γ and a table T

if $i < \gamma$ **then**
  $e \leftarrow h_i$, where $(q_i, h_i) \in T$
  Append $\{q_i, e\}$ to $\tilde{T}$
else if $i = \gamma$ **then**
  $e \xleftarrow{\$} \mathcal{C} \setminus \{h_\gamma\}$, where $(q_\gamma, h_\gamma) \in T$
  Append $\{q_i, e\}$ to $\tilde{T}$
else
  if $\exists h_i, \{q_i, h_i\} \in \tilde{T}$ **then**
    $e \leftarrow h_i$
  else
    $e \xleftarrow{\$} \mathcal{C}$
    Append $\{q_i, e\}$ to $\tilde{T}$
  end if
end if
return e