Confirmer Signature Schemes Secure against Adaptive Adversaries

(Extended Abstract)

Jan Camenisch
IBM Research
Zurich Research Laboratory
CH–8803 Rüschlikon
jca@zurich.ibm.com

Markus Michels
Entrust Technologies (Switzerland)
Glatt Tower
CH–8301 Glattzentrum
Markus.Michels@entrust.com

Abstract. The main difference between confirmer signatures and ordinary digital signatures is that a confirmer signature can be verified only with the assistance of a semitrusted third party, the confirmer. Additionally, the confirmer can selectively convert single confirmer signatures into ordinary signatures.
This paper points out that previous models for confirmer signature schemes are too restricted to address the case where several signers share the same confirmer. More seriously, we show that various proposed schemes (some of which are provably secure in these restricted models) are vulnerable to an adaptive *signature-transformation* attack. We define a new stronger model that covers this kind of attack and provide a generic solution based on any secure ordinary signature scheme and public key encryption scheme. We also exhibit a concrete instance thereof.

1 Introduction

To limit the information dispersed by digital signatures, Chaum and van Antwerpen introduced the concept of *undeniable* signatures [10]. Undeniable signatures can only be verified with the help of the original signer. Of course, the signer should be able to deny invalid signatures but must not be able to deny valid signatures. Thereby the signer is able to control who gets to know the validity of a signature. To overcome this concept's shortcoming that signers might be unavailable or unwilling to cooperate and hence signatures would no longer be verifiable, Chaum suggested the notion of *confirmer signatures* [9]. Here, the ability to verify/deny signatures is transferred to a semitrusted third party, the confirmer. The confirmer is also given the power to convert a confirmer signature into an ordinary (i.e., publicly verifiable) signature. Of course, the confirmer should not be involved in the signing process. It is understood that the confirmer follows some policy for deciding to whom he confirms signatures or which signatures he can convert and under which circumstances (e.g., such a policy could be included in the signed message). For instance, a policy could state that confirmation is only allowed during a certain time period, only to a certain group of people, or simply that the confirmer must log all requests.

Chaum also presented a concrete scheme but neither a formal model nor a proof of security [9]. Later, Okamoto presented a formal model and proved

that confirmer signature schemes are equivalent to public-key encryption [23]. Okamoto further presented a practical solution. However, Okamoto's model explicitly enables not only the confirmer but also the signer to assist in verification of a confirmer signature. A drawback of this approach is that a coercer could force the signer to cooperate in confirming or denying a signature. Although a signer is in principle always able to prove that a confirmer signature he generated is valid (e.g., by proving knowledge of all inputs to the signing algorithm), the signer can always claim that he did not generate an alleged confirmer signature and thus is unable to prove anything, if confirmer signatures are "invisible", i.e., if it is undecidable for everybody apart from the confirmer whether a confirmer signature is valid or not. This coercer problem is (partially) overcome in the model of Michels and Stadler [21], which does not explicitly enable the signer to deny invalid signatures. They also showed that Okamoto's practical scheme is insecure because the confirmer can fake signatures. Moreover, they proposed new schemes and proved them secure in their model. Finally, all realizations proposed so far [9, 11, 21–23] incorporate the feature that the confirmer could convert confirmer signatures into ordinary but proprietary signatures (i.e., not standard signatures such as RSA PKCS#1 or DSS). However, this convertibility is not included in any of their models and it is hence uncertain whether the schemes remain secure if this feature is activated.

The contribution of this paper is to point out that various proposed confirmer schemes are insecure when many signers share the same confirmer. The latter seems to be natural in an e-commerce environment where playing the role of a confirmer is offered as a trusted third party service and signers decide on a per-signature basis which confirmer to use (possibly considering requirements from the signature receiver). More precisely, these schemes are vulnerable to an adaptive *signature-transformation* attack, where the attacker transforms a confirmer signature with respect to given signing keys into a confirmer signature with respect to other signing keys such that the resulting confirmer signature is valid only if the original signature is valid. With this new signature the attacker can enter the confirmation protocol thus circumvent the policy of the original signature. For instance, such attacks are possible against the schemes in [21] that were proved secure with respect to the model given there and applies also to some of the schemes presented in [9, 23]. We argue that the formal models [21, 23] proposed so far are too restrictive, e.g., as this kind of attack is not incorporated.

This paper exhibits a new model that fully incorporates adaptive adversaries. The model also explicitly includes the convertibility of confirmer signatures into ordinary signature schemes and excludes the signer's ability to deny invalid signatures. We present a generic solution based on any signature scheme that is secure against an adaptive chosen-message attack and on any encryption scheme that is secure against an adaptive chosen-ciphertext attack and prove its security in our model. This solution enjoys *perfect convertibility*, i.e., converted signatures are signatures with respect to the signature scheme we use as a building block. This property is unmet by all previously proposed schemes. We also provide a concrete instance based on any deterministic RSA signature scheme and the

Cramer-Shoup encryption scheme. An adaption to other signature schemes such as DSS is easily possible using techniques from [1]. Moreover, we outline how the scheme of Michels and Stadler can be adapted to be secure in our model and how scenarios such as fair contract signing and verifiable signature sharing can be addressed.

2 Confirmer Signature Model

This section provides a formal definition of confirmer signatures. After having defined our model, we discuss the differences to previously suggested models in detail and point out why various previously proposed schemes fail in our model.

2.1 Formal Model

Definition 1. *The players in a confirmer signature scheme are signers S, confirmers C, and verifiers V. A confirmer signature scheme consists of the following procedures:*

Key generation: *Let $CKGS(1^\ell) \to (x_S, y_S)$ and $CKGC(1^\ell) \to (x_C, y_C)$ be two probabilistic algorithms. The parameter ℓ is a security parameter, (x_S, y_S) is a secret/public key pair for the signer, and (x_C, y_C) is a secret/public key pair for the confirmer.*

Signing: *A probabilistic signature generation algorithm $CSig(m, x_S, y_S, y_C) \to \sigma$ for signing a message $m \in \{0,1\}^*$.*

Confirmation and disavowal: *A signature verification protocol $(CVerC, CVerV)$ between a confirmer and a verifier. The private input of the confirmer is x_C and their common input consists of m, σ, y_S, and y_C. The output of the verifier is either 1 (true) and 0 (false).*

Selective convertibility: *An algorithm $CConv(m, \sigma, y_S, x_C, y_C) \to s$ that allows a confirmer to turn a confirmer signature σ into an ordinary signature. If the conversion fails, the algorithm's output is $\perp$.*

Signature verification (ordinary): *An algorithm $COVer(m, s, y_S) \to \{0,1\}$ that allows everybody to verify signatures and takes as input a message m, a signature s, and the public key y_S of the signer.*

Before formally stating the security requirements, we try to describe the intuition behind them. Correctness and validity of confirmation/disavowal, and correctness of conversion are obvious. Security for the signer guarantees that confirmer signatures as well as converted signatures are unforgeable under an adaptive chosen-message attack (cf. [20]). Security for the confirmer/invisibility of signatures guarantees that the scheme is secure for the confirmer against adaptive chosen-confirmer-signature attacks (this is similar to security against chosen-ciphertext attacks for encryption schemes, in fact, CSig can be regarded as an encryption scheme for a single bits). This requirement also assures that no one apart from the confirmer can distinguish between valid and invalid confirmer

signatures. This ensures for instance that the signer is not coercible. Finally, non-transferability says that one cannot get more information out of the confirmation/disavowal protocol than whether a signature is valid or not.

By $\{A(u)\}$ we denote the set of all possible output values of a probabilistic algorithm A when input u.

Correctness of confirmation/disavowal: *If the confirmer and the verifier are honest, then for all ℓ, all $(x_S, y_S) \in \{CKGS(1^\ell)\}$, all $(x_C, y_C) \in \{CKGC(1^\ell)\}$, all $m \in \{0,1\}^*$, and all $\sigma \in \{0,1\}^*$,*

$$CVerV_{CVerC}(m, \sigma, y_S, y_C) = \begin{cases} 1 & \text{if } \sigma \in \{CSig(m, x_S, y_S, y_C)\} \\ 0 & \text{otherwise .} \end{cases}$$

Validity of confirmation/disavowal: *For all $CVerC^*$, all sufficiently large ℓ, all $(x_S, y_S) \in \{CKGS(1^\ell)\}$, all $(x_C, y_C) \in \{CKGC(1^\ell)\}$, all $m \in \{0,1\}^*$, all $\sigma \in \{0,1\}^*$, and for every polynomial $p(\cdot)$ we require that*

$$Prob\,[CVerV_{CVerC^*}(m, \sigma, y_S, y_C) = 0] < 1/p(\ell)$$

if $\sigma \in \{CSig(m, x_S, y_S, y_C)\}$ and

$$Prob\,[CVerV_{CVerC^*}(m, \sigma, y_S, y_C) = 1] < 1/p(\ell)$$

otherwise. The probability is taken over the coin tosses of $CVerV$ and $CVerC^*$.

Correctness of conversion: *For all ℓ, all $(x_S, y_S) \in \{CKGS(1^\ell)\}$, all $(x_C, y_C) \in \{CKGC(1^\ell)\}$, all $m \in \{0,1\}^*$, and for all $\sigma \in \{CSig(m, x_S, y_S, y_C)\}$, it holds that $COVer(m, CConv(m, \sigma, y_S, x_C, y_C), y_S) = 1$.*

Security for the signer: *Consider the following game against an adversary A. First the key generators for the signer and the confirmer are run on input 1^ℓ. Then A is given as input the public key of the signer and the confirmer, y_S and y_C, and the secret key x_C of the confirmer. A is further allowed oracle access to the signer (i.e., it may ask confirmer signatures of polynomially many messages $\{m_i\}$). Finally, A halts and outputs a pair of strings (m, u) where $m \neq m_i$ for all i. Then, for all such A and all sufficiently large ℓ we require that A's output satisfies $COVer(m, u, y_S) = 1$ with negligible probability only. The probability is taken over the coin tosses of the signer, A, and the key generators. (Note that the adversary can convert confirmer signatures itself as it is given x_C.)*

Security for the confirmer / Invisibility of Signatures: *Consider the following game against an adversary A. First the key generators for the signer and the confirmer are run on input 1^ℓ. The adversary is given the public keys of the signer and the confirmer, in addition to the secret key of the signer. Then the adversary can make arbitrary oracle queries to the confirmer via $CVerC$ and $CConv$. For doing this, the adversary is allowed at anytime (and repeatedly) to create additional signature-key pairs $(x_{S'}, y_{S'})$ (not necessarily by running the key generator) and to interact with the confirmer with respect to these keys. Then, the adversary has to present two messages $m_1, m_2 \in \{0,1\}^*$.*

After that we flip a fair coin. If the result is heads, the adversary is given $\sigma = CSig(m_1, x_S, y_S, y_C)$*, if it is tails, the adversary is given a string* $\sigma = CSig(m_2, x_S, y_S, y_C)$*. Now the adversary is again allowed to query the signer and the confirmer except that* σ *is not allowed in any of these queries. Finally, the adversary must output* 0 *or* 1*. We require that for all such adversaries, all polynomials* $p(\cdot)$*, and all sufficiently large* ℓ*, the probability that the adversary's output equals our coin flip is smaller than* $1/2 + 1/p(\ell)$*, where the probability is taken over the coin tosses of the signer, the confirmer, and the key generators.*

Non-transferability of verification/disavowal: *Consider the following two games involving the adversary, a signer, a confirmer, and a simulator:*

Game 1. The adversary is given the public keys y_S *and* y_C *of the signer and the confirmer. Then it can make arbitrary oracle queries to both of them via CSig, CVerC, and CConv. (Again the adversary is allowed at any time to create its own key pairs* $(x_{S'}, y_{S'})$ *and run, e.g., CSig with these keys, and then interact with the confirmer with respect to these keys as well.) Then the adversary must present two strings,* m *and* σ *for which it wishes to carry out the protocol* $(CVerC, CVerV)$ *with the confirmer. Next the confirmer and the adversary carry out this protocol with common input* (m, σ, y_S, y_C)*. The confirmer's secret input will be* x_C*. In parallel, the adversary is allowed to make arbitrary queries to the signer and confirmer. Eventually, the adversary stops producing an output.*

Game 2. This game is the same as Game 1 with the difference that when it comes to the interaction with the confirmer on m *and* σ *the simulator is plugged in the place of the confirmer. However, in all other interactions with the adversary the real confirmer or the real signer speak with the adversary. The simulator is not given the secret key of the confirmer, but it is allowed a single call to an oracle that tells it whether the strings* m *and* σ *produced by the adversary are a valid confirmer signature w.r.t.* y_S *and* y_C*.*

Now we require that for every adversary there exists a simulator such that for all sufficiently large ℓ*, all* $(x_S, y_S) \in \{CKGS(1^\ell)\}$*, and all* $(x_C, y_C) \in \{CKGC(1^\ell)\}$*, the outputs of the adversary when playing Game 1 and Game 2 are indistinguishable. In other words, there must exist a simulator such that the adversary cannot distinguish whether he is playing Game 1 or 2.*

We call a confirmer signature scheme *perfect convertible* with respect to some (ordinary) signature scheme if converted confirmer signatures are valid signatures with respect to this signature scheme.

Throughout the paper we assume that the policy stating the circumstances under which the confirmer is allowed to confirm/disavow a confirmer signature is part of the actual message and that he refuses cooperation whenever the policy requires so. This is sufficient to ensure that verifiers cannot circumvent a policy.

Schemes according to our definition are *separable*, i.e., all parties can run their key generation algorithms independent of each other (cf. [6]). This enables signers to choose a confirmer on a per signature basis at signing time.

Remark 1. One could easily add a protocol between a confirmer signature recipient and the signer in which the signer proves to the recipient that a confirmer signature *just generated* is valid. The only modification to our model would be that one would have to add a security requirement for this protocol that is similar to the one of *non-transferability of verification/disavowal* for $(CVerC, CVerV)$. Furthermore, the adversary has to be allowed to use this new protocol in the games defined in *security for the signer* and *non-transferability of verification/disavowal*.

2.2 Comparison with Previous Formal Models

Let us point out the differences between our model and the previous formal models [21, 23].

As mentioned in the introduction, Okamoto's model enables the signer to confirm and deny signatures, which makes the signer vulnerable to a coercer that forces him to confirm or deny a signature. The model does not include selective conversion. Moreover, his model defines a weaker notion of security of the confirmer: the adversary knowing the signer's secret key wins the game only if he is able to behave like the confirmer, i.e., to confirm and disavowal signatures, but does not win the game if he can distinguish between two confirmer signatures (or between a valid and an invalid confirmer signature). The crucial difference, however, lies in the definition of invisibility and untransferability, where the adversary has access only to the confirmation and disavowal protocol run with the *true signer*, but *not* with the confirmer. Thus it does not cover adaptive attacks against the confirmer. For instance, the signature transformation attack mentioned below is not captured by this model. In fact, one can construct a scheme that is secure in Okamoto's model but is vulnerable to this signature transformation attack. Such a scheme is obtained from one of the schemes in [21] by having the signer choose an encryption public key and then appending to the signature an encryption of all random choices made by the signer in the signing protocol under this public key (this encryption also must be signed together with the message). This will allow the signer to confirm/disavow signatures as required in Okamoto's model.

The model by Michels and Stadler does not explicitly enable the signer to confirm and deny signatures, but it does not exclude it either. In particular, the security for the confirmer (where the adversary gets the signer's secret key) as well as the selective conversion are not included. Their definition of invisibility allows the adversary only to query the confirmer with respect to a certain signer and is not given the signer's secret key, i.e., they allow only a very restricted kind of adaptive attack. This model is realistic only if there is a single signer that is furthermore assumed to be honest. However, if several signer are allowed and they are not all assumed to be honest, then their schemes are vulnerable to the signature transformation attack as described in the next paragraph.

2.3 Adaptive Signature-Transformation Attacks

This paragraph points out that the previously suggested schemes [9, 21, 23] are vulnerable to an adaptive attack and are indeed insecure in our model. Before describing this attack, we note that the scheme proposed in [22] is not secure in our model because is has the property that given a signature and two different messages it's publically verifiable w.r.t. which message the signature is potentially valid. Due to this property the invisibility requirement in our model cannot be satisfied. Furthermore, the scheme presented in [11] is insecure in *all* models, i.e., even against non-adaptive attackers (see Appendix A).

We first show that the proof-based scheme by Michels and Stadler [21, Section 5.2], which was proved secure in their model, is vulnerable to a so-called adaptive signature-transformation attack that exploits the malleability of the used building block. The practical scheme by Okamoto [23], with or without the heuristic fix of another vulnerability suggested in [21], as well as Chaum's scheme [9] are vulnerable to a similar attack. We omit the details regarding those schemes here.

Let us first recall the scheme by Michels and Stadler. It uses as building blocks so-called *proof-based signature schemes* (an example is Schnorr's signature scheme [26]) and *confirmer commitments*. For simplicity, let us use the confirmer commitment provided in [21, Section 4.2] and the Schnorr signature scheme [26] in the following. With these choices the public key of the signer is a group $G = \langle g \rangle$, a prime $q = |G|$, and $y \in G$. The signer's secret key is $x = \log_g y$. The confirmer's public key is $H = \langle h \rangle$, a prime $p = |H|$, and $z \in H$. The confirmer's secret key is $u = \log_h z$. Furthermore, a suitable hash function $\mathcal{H}$ is publicly known. The signer can issue a confirmer signature on m as follows.

1. $r_1 \in \mathbb{Z}_q$, $r_2 \in \mathbb{Z}_p$, $t := g^{r_1}$, $d := (d_1, d_2) := (z^{r_2}, h^{r_2 + \mathcal{H}(t,m)})$,
2. $c := \mathcal{H}(d)$, and $s := r_1 - cx \bmod q$.

The confirmer signature is $(t, (d_1, d_2), s)$. The confirmer can tell whether a given confirmer signature $(t, (d_1, d_2), s)$ is valid by checking if $d_2 / d_1^{1/u} \stackrel{?}{=} h^{\mathcal{H}(t,m)}$ and $y^{\mathcal{H}(d)} g^s \stackrel{?}{=} t$ hold. We refer to [21] for the confirmation/disavowal protocol.

Now we are ready to describe the signature transformation attack. We are given an alleged confirmer signature $(t, (d_1, d_2), s)$ on m w.r.t. a signer's public key (G, g, q, y). Furthermore, assume that the confirmer is not allowed to tell us whether this particular signature is a valid. The following attack will allow us to transform the signature into another signature that is independent from $(t, (d_1, d_2), s)$. To do so, we choose our own signing public and secret keys $\tilde{G} = \langle \tilde{g} \rangle$ with $|\tilde{G}| = \tilde{q}$, $\tilde{y} = \tilde{g}^{\tilde{x}}$. Then we choose a random message $\tilde{m}$ and

1. $\tilde{r}_1 \in \mathbb{Z}_{\tilde{q}}$, $\tilde{r}_2 \in \mathbb{Z}_p$, $\tilde{t} := \tilde{g}^{\tilde{r}_1}$, $\tilde{d} := (\tilde{d}_1, \tilde{d}_2) := (d_1 z^{\tilde{r}_2}, d_2 h^{\tilde{r}_2 + \mathcal{H}(\tilde{t},\tilde{m}) - \mathcal{H}(t,m)})$,
2. $\tilde{c} := \mathcal{H}(\tilde{d})$, and $\tilde{s} := \tilde{r}_1 - \tilde{c}\tilde{x} \bmod \tilde{q}$

and get the new confirmer signature $(\tilde{t}, (\tilde{d}_1, \tilde{d}_2), \tilde{s})$. This confirmer signature is valid if and only if the original confirmer signature $(t, (d_1, d_2), s)$ is valid. Furthermore, if the original confirmer signature is valid, then the new confirmer

signature is indistinguishable from a confirmer signature made using the real signing algorithm with our public key. Hence we can simply feed the signature $(\tilde{t}, (\tilde{d}_1, \tilde{d}_2), \tilde{s})$ to the confirmer and he will tell in fact whether $(t, (d_1, d_2), s)$ is valid. This attack breaks the invisibility property, and it is possible because the confirmer commitment is malleable. Note that the definition of security for confirmer commitments in [21] does not consider adaptive adversaries.

A variant of this attack works even if the used confirmer commitment is non-malleable: After the attacker has obtained the confirmer signature (t, d, s) on m w.r.t. a signer's public key (G, g, q, y), he computes a new public key $(G, \tilde{g}, q, \tilde{y})$ by picking $\tilde{r}_1, \tilde{x} \in \mathbb{Z}_q$ and computing $\tilde{g} := t^{1/\tilde{r}_1}$ and $\tilde{y} := \tilde{g}^{\tilde{x}}$. Now $(t, d, (\tilde{r}_1 - \mathcal{H}(d)\tilde{x} \bmod q))$ will be a valid confirmer signature on m w.r.t. the signer's public key $(G, \tilde{g}, q, \tilde{y})$ if and only if (t, d, s) is a valid w.r.t. (G, g, q, y). In a similar way as above this attack breaks the invisibility property. The second scheme proposed in [21] is also vulnerable to this attack. However, this kind of attack can be easily countermeasured by adding the signer's public key to the input of the confirmer commitment.

3 A Generic Realization of Confirmer Signature Schemes

This section presents a generic confirmer signature scheme, proves its security, and discusses its implications. As we will see in the next section, this generic scheme has concrete instances that are quite efficient.

Let $SIG = (SKG, Sig, Ver)$ denote a signature scheme, where SKG is the key-generation algorithm (which on input 1^ℓ outputs a key pair (x, y)), Sig is the signing algorithm (which on input of a secret key x, the corresponding public key y, and a message $m \in \{0,1\}^*$ outputs a signature s on m), and Ver is the verification algorithm (which on input of a message m, an alleged signature s, and public key y outputs 1 if and only if s is a signature on m with respect to y). Moreover, let $ENC = (EKG, Enc, Dec)$ denote a public key encryption scheme. On input of a security parameter, EKG outputs a key pair (x', y'). On input of a public key y' and a message m', Enc outputs a ciphertext c. On input of the ciphertext c of the message m', the secret key x', and the public key y', Dec outputs m' if c is valid and $\perp$ otherwise.

Given a suitable signature scheme $SIG = (SKG, Sig, Ver)$ and a suitable encryption scheme $ENC = (EKG, Enc, Dec)$, a confirmer signature scheme can be constructed as follows. We will later see what suitable means.

1. The respective key generators are chosen as $CKGS(1^\ell) \mathrel{\widehat{=}} SKG(1^\ell)$ and $CKGC(1^\ell) \mathrel{\widehat{=}} EKG(1^\ell)$.
2. The signer signs a message $m \in \{0,1\}^*$ by computing $s := Sig(x_S, y_S, m)$ and $e := Enc(y_C, s)$. The confirmer signature on m is given by e.
3. The confirmation and disavowal protocol $(CVerC, CVerV)$ between the confirmer and a verifier is done as follows: Given an alleged confirmer signature e and a message m, the confirmer decrypts e to get $\hat{s} := Dec(e, x_C, y_C)$. If $Ver(m, \hat{s}, y_S) = 1$, then the confirmer tells the verifier that the confirmer

signature is valid and shows this by proving in *concurrent* zero-knowledge that he knows values α and β such that "β is the secret key corresponding to y_C AND $\alpha = Dec(e, \beta, y_C)$ AND $Ver(m, \alpha, y_S) = 1$." Otherwise, the confirmer tells the verifier that the confirmer signature is invalid and proves in *concurrent* zero-knowledge that he knows values α and β such that "β is the secret key corresponding to y_C AND (($\alpha = Dec(e, \beta, y_C)$ AND $Ver(m, \alpha, y_S) = 0$) OR decryption fails)."

4. The selective conversion algorithm $CConv(m, e, y_S, x_C, y_C)$ outputs $Dec(e, x_C, y_C)$, provided $Ver(m, Dec(e, x_C, y_C), y_S) = 1$, and $\perp$ otherwise.
5. The public verification algorithm for converted signatures is defined as $COVer(m, s, y_S) \mathrel{\widehat{=}} Ver(m, s, y_S)$.

Theorem 1. *If SIG is existentially unforgeable under an adaptive chosen-message attack and ENC is secure against adaptive chosen-ciphertext attacks, then the above construction constitutes a secure confirmer signature scheme with perfect conversion.*

Proof: [Sketch] The properties *correctness of confirmation/disavowal*, *validity of confirmation/disavowal*, and *correctness of conversion* are obviously satisfied. Let us consider the remaining properties.

Security for the signer: We show that if there is an adversary A that can forge a confirmer signature, then A could be used to forge signatures of the signature scheme *SIG* in an adaptive chosen-message attack: The messages m_i that are queried by A are simply forwarded to the signing oracle of the underlying signature scheme *SIG* and then the result is encrypted using *ENC*. If A is able to produce a valid confirmer signature to any message that is not in the set of queried messages, we can convert this confirmer signature into a valid ordinary signature by the conversion algorithm. If A is able to compute a valid signature to any message that is not in the set of messages previously queried, we are already done. Both cases contradict the security of *SIG*.

Security for the confirmer/Invisibility of signatures: We show that if there exists an adversary A that can violate this property, then the encryption scheme *ENC* is not secure against adaptive chosen-ciphertext attacks: When getting A's request for confirmation/disavowal of a message m and an alleged confirmer signature e, we forward e to the decryption oracle of the underlying encryption scheme and obtain s. If s is an (ordinary) signature of m, then we tell A that the confirmer signature is valid and carry out the proof that this is indeed the case. Of course, we cannot carry out the real protocol, but as it is required to be concurrent zero-knowledge, there exists a simulator for it which we can use. The case where s is not a valid signature is similar. If A requests the conversion of m and e, we forward e to the decryption oracle, get s, and then we output s if s is a valid ordinary signature on m, or $\perp$ otherwise.

When it comes to the point where A presents the "test messages" m_1 and m_2, we produce signatures of them, i.e., s_1 and s_2, and present these as "test messages" to the game in the underlying encryption scheme. Then we forward the encryption we get as challenges from the underlying game to A as a

challenge. The following queries of A are handled as before. When A eventually halts and outputs 0 or 1, we forward this output as an answer in the game against the underlying encryption scheme. This concludes the reduction.

Non-transferability of verification/disavowal: This property follows in a straightforward manner from the concurrent zero-knowledge property of the proofs in the confirmation/disavowal protocol.

□

Corollary 1. *I. If trapdoor one-way permutations exist then there exists a secure confirmer signature scheme. II. A secure confirmer signature scheme exists if and only if a public key encryption scheme secure against adaptive chosen-ciphertext attacks exists (cf. [23, Theorem 3]).*

Proof: Part I. The existence of trapdoor one-way permutations implies a secure signature scheme and an encryption scheme secure against adaptive chosen-ciphertext attacks [3, 16, 20, 25]. Due to Theorem 1, this is sufficient for a secure confirmer signature scheme. Part II. On the one hand, an encryption scheme for encrypting a single bit follows from a secure confirmer signature scheme (cf. [23]). Let the public key of the encryption scheme be the public key of the confirmer. To encrypt, one chooses a signer's key pair and then a 0 is encrypted by issuing a valid confirmer signature on a randomly chosen message and a 1 is encrypted by issuing a simulated (invalid) confirmer signature on a randomly chosen message. On the other hand, if a secure public key encryption scheme exists then there exist one-way functions and hence a signature scheme secure against adaptive chosen-message attacks [20, 25]. Due to Theorem 1, this is sufficient for a secure confirmer signature scheme. □

Remark 2. The generic confirmer signature scheme exhibited in this section provides perfect convertibility with respect to the signature scheme *SIG*.

Remark 3. The described generic confirmer signature scheme has some similarities to the generic scheme due to Okamoto[23]. However, as Okamoto's model requires the signer to have the ability to deny invalid confirmer signature scheme this scheme cannot satisfy the invisibility property as stated above. Whereas Okamoto's generic scheme is a theoretical construction requiring general zero-knowledge proofs for confirmation and disavowal, our scheme has concrete instances with quite efficient protocols for confirmation and disavowal.

4 An Instance Providing Perfect Conversion of Signatures

This section provides an instance based on an arbitrary deterministic RSA signature scheme [24] and the Cramer–Shoup encryption scheme [13]. Instances for other signature schemes such as DSS or Schnorr can be realized similarly using the signature reduction techniques from [1].

4.1 Notation

We use notation from [6, 7] for the various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For instance,

$$PK\{(\alpha,\beta,\gamma) : y = g^{\alpha}h^{\beta} \ \wedge \ \tilde{y} = \tilde{g}^{\alpha}\tilde{h}^{\gamma} \ \wedge \ (u \leq \alpha \leq v)\} \ ,$$

denotes a "*zero-knowledge* P*roof of* K*nowledge of integers* α*,* β*, and* γ *such that* $y = g^{\alpha}h^{\beta}$ *and* $\tilde{y} = \tilde{g}^{\alpha}\tilde{h}^{\gamma}$ *holds, where* $v < \alpha < u$," where $y, g, h, \tilde{y}, \tilde{g}$, and $\tilde{h}$ are elements of some groups $G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. The convention is that Greek letters denote the knowledge proved, whereas all other parameters are known to the verifier. The scheme presented in this section uses proofs of knowledge of double discrete logarithms and of roots of discrete logarithms [7, 27] and proofs that a discrete logarithm lies in an interval [5, 12], e.g., $(u \leq \log_g y \leq v)$. These protocols are 3-move zero-knowledge proofs of knowledge with binary challenges.

An important variant of such protocols are *concurrent* zero-knowledge proofs (of knowledge). They are characterized by remaining zero-knowledge even if several instances of the same protocol are run arbitrarily interleaved [14, 15, 17]. Damgård [15] shows that 3-move proofs (this includes all protocols considered in this paper) can easily be made concurrent zero-knowledge in many practical scenarios. We denote the resulting protocols by, e.g., *CZK-PK*$\{\alpha : y = g^{\alpha}\}$

4.2 Description of the Scheme

We review both schemes we use as building block briefly and then describe the resulting confirmer signature scheme.

Let (n, e) be an RSA public key of a signer and $Pad_S(\cdot) : \{0,1\}^* \to \{1, \ldots, n\}$ be some padding function. To sign a message $m \in \{0,1\}^*$, the signer computes $s := Pad_S(m)^{1/e} \bmod n$. To verify a signature s on m, one checks whether $s^e \equiv Pad_S(m) \pmod{n}$. If the padding function is assumed to be a truly random function, the system is secure against adaptively chosen-message attacks under the RSA assumption [4].

The Cramer–Shoup encryption scheme works over some group H of (large) prime order q of which two generators h_1 and h_2 are known. The secret key consists of five elements $x_1, \ldots, x_5 \in_R \mathbb{Z}_q$ and the public key (y_1, y_2, y_3) is computed as $y_1 := h_1^{x_1}h_2^{x_2}$, $y_2 := h_1^{x_3}h_2^{x_4}$, and $y_3 := h_1^{x_5}$. Encryption of a message $m \in H$ is done by choosing a random $r \in_R \mathbb{Z}_q$ and computing $c_1 := h_1^r$, $c_2 := h_2^r$, $c_3 := y_3^r m$, and $c_4 := y_1^r y_2^{r\mathcal{H}(c_1,c_2,c_3)}$. Decryption of a tuple $(c_1, c_2, c_3, c_4) \in H^4$ is done by computing $u := \mathcal{H}(c_1, c_2, c_3)$ and checking whether $c_1^{x_1+x_3u}c_2^{x_2+x_4u} = c_4$. If this condition does not hold, the decryption algorithm outputs $\perp$. Otherwise, it computes $m' := c_3/c_1^{x_5}$ and outputs m'. Provided the decision Diffie–Hellman assumption holds in H and the hash function $\mathcal{H}$ used is chosen collision resistant, the system is secure against adaptive chosen-ciphertext attacks [13].

We are now ready to present the different procedures of the confirmer signature scheme.

Key Generation: *CKGS*: The signer chooses an RSA public key (n, e). Furthermore, the signer also publishes a group $G = \langle g_1 \rangle = \langle g_2 \rangle$ of order n. *CKGC*: The confirmer chooses sufficiently large primes q and $p = 2q + 1$ and two elements h_1 and h_2 from $\mathbb{Z}_p^\star$ such that $\left(\frac{h_1}{p}\right) = \left(\frac{h_2}{p}\right) = 1$ and $\log_{h_1} h_2$ is unknown. Furthermore, the confirmer publishes a group $\tilde{H} = \langle \tilde{h}_1 \rangle$ of order p. This group is required for the proofs in the confirmation/disavowal protocol. A collision resistant hash function $\mathcal{H}$ is fixed.

Signing: We assume $n < p/2$. (The case $p/2 < n$ can be handled by splitting the signature into two or more parts before encryption. We refer to the forthcoming full version of the paper for details.) To sign a message $m \in \{0,1\}^*$, the signer computes $\hat{s} := Pad_S(m)^{1/e} \bmod n$, sets $s := \hat{s}$ if $\left(\frac{\hat{s}}{p}\right) = 1$ and $s := p - \hat{s}$ otherwise (hence $\left(\frac{s}{p}\right) = 1$). The signer encrypts s by choosing a random $r \in_R \mathbb{Z}_q$ and computing $c_1 := h_1^r$, $c_2 := h_2^r$, $c_3 := y_3^r s$, and $c_4 := y_1^r y_2^{r\mathcal{H}(c_1,c_2,c_3)}$. The confirmer signature on m is $\sigma := (c_1, c_2, c_3, c_4)$.

Confirmation and disavowal: The verifier chooses $\tilde{h}_2 \in_R \tilde{H}$ and $h_3 \in_R H$. Upon receipt of a request $m \in \{0,1\}^*$, $(n, e) \in \{0,1\}^* \times \mathbb{Z}_n$, $\sigma = (c_1, c_2, c_3, c_4) \in H^4$, and $(\tilde{h}_2, h_3) \in \tilde{H} \times H$ from a verifier the confirmer first decrypts (c_1, c_2, c_3, c_4) and gets a value $\hat{s}$ if decryption does not fail. If $0 < \hat{s} < n$ he sets $s := \hat{s}$ and $s := p - \hat{s}$ otherwise. If $0 < s < n$ and $s^e \equiv Pad_S(m) \pmod n$ holds, the confirmer tells the verifier that the confirmer signature σ is valid and otherwise that it is not valid.

If σ is valid (confirmation): The confirmer computes commitments $C_1 := g_1^s g_2^{v_1}$ and $C_2 := \tilde{h}_1^{\hat{s}} \tilde{h}_2^{v_2}$ with $v_1 \in_R \mathbb{Z}_n$ and $v_2 \in_R \mathbb{Z}_p$ and sends C_1 and C_2 to the verifier. Then, confirmer and verifier carry out the following protocol:

$$
\begin{aligned}
&CZK\text{–}PK\{(\alpha, \beta, \gamma, \varepsilon, \vartheta, \lambda, \rho, \xi, \alpha, \nu_1, \ldots, \nu_6) : \\
&\qquad y_1 = h_1^\gamma h_2^\beta \ \wedge\ y_2 = h_1^\lambda h_2^\rho \ \wedge\ y_3 = h_1^\xi \ \wedge \\
&\qquad \tilde{h}_1^{c_3} = C_2^{c_1^\xi} \tilde{h}_2^{\nu_1} \ \wedge\ c_4 = c_1^\gamma (c_1^{\mathcal{H}(c_1,c_2,c_3)})^\lambda c_2^\beta (c_2^{\mathcal{H}(c_1,c_2,c_3)})^\rho \ \wedge \\
&\qquad \Big(\big(C_2 = \tilde{h}_1^\varepsilon \tilde{h}_2^{\nu_2} \ \wedge\ C_1 = g_1^\varepsilon g_2^{\nu_3} \ \wedge\ (1 \le \varepsilon \le n-1) \big) \vee \\
&\qquad \big(C_2 = (1/\tilde{h}_1)^\vartheta \tilde{h}_2^{\nu_4} \ \wedge\ C_1 = g_1^\vartheta g_2^{\nu_5} \ \wedge\ (1 \le \vartheta \le n-1) \big) \Big) \wedge \\
&\qquad\qquad C_1 = g_1^\alpha g_2^{\nu_6} \ \wedge\ g_1^{Pad_S(m)} = g_1^{\alpha^e} \}.
\end{aligned}
$$

With this protocol the confirmer convinces the verifier that decryption was successful and that either the decrypted value or p minus the decrypted value are a valid RSA signature with respect to m, e, n, and Pad_S. We refer the reader to the full paper for the protocol in all its details.

If σ is not valid (disavowal): If decryption failed, the confirmer chooses $\hat{s} \in_R \mathbb{Z}_p$ and $s \in_R \mathbb{Z}_N$ such that $\left(\frac{\hat{s}}{p}\right) = 1$. Then he computes the following commitments $C_1 := g_1^s g_2^{v_1}$, $C_2 := \tilde{h}_1^{\hat{s}} \tilde{h}_2^{v_2}$, $C_3 := g_1^{s^e} g_2^{v_3}$, $C_4 := h_3^{v_4} c_1^{x_1 + x_2 \mathcal{H}(c_1,c_2,c_3)} c_2^{x_3 + x_4 \mathcal{H}(c_1,c_2,c_3)}$, and $C_5 := \tilde{h}_1^{h_3^{v_4}} \tilde{h}_2^{v_5}$ with $v_1, v_3 \in_R \mathbb{Z}_n$, $v_2, v_5 \in_R \mathbb{Z}_p$, and $v_4 \in_R \mathbb{Z}_q$. He sends $(C_1, C_2, C_3, C_4, C_5)$ to the verifier.

Confirmer and verifier carry out the following protocol:

$$
\begin{aligned}
&\mathit{CZK\text{–}PK}\Big\{(\gamma,\beta,\lambda,\rho,\xi,\delta,\kappa,\alpha_1,\alpha_2,\alpha_3,\nu_1,\ldots,\nu_{14}):\\
&\qquad y_1 = h_1^{\gamma}h_2^{\beta} \ \wedge\ y_2 = h_1^{\lambda}h_2^{\rho} \ \wedge\ y_3 = h_1^{\xi} \ \wedge\\
&\Big((C_4 = h_3^{\delta}c_1^{\gamma}(c_1^{\mathcal{H}(c_1,c_2,c_3)})^{\lambda}c_2^{\beta}(c_2^{\mathcal{H}(c_1,c_2,c_3)})^{\rho} \ \wedge\ C_5 = \tilde{h}_1^{h_3^{\delta}}\tilde{h}_2^{\nu_1} \ \wedge\\
&\qquad \tilde{h}_1 = (\tilde{h}_1^{C_4/c_4}/C_5)^{\kappa}(1/\tilde{h}_2)^{\nu_2}\Big) \ \vee\\
&\Big(\tilde{h}_1^{c_3} = C_2^{c_1^{\xi}}\tilde{h}_2^{\nu_3} \ \wedge\ c_4 = c_1^{\gamma}(c_1^{\mathcal{H}(c_1,c_2,c_3)})^{\lambda}c_2^{\beta}(c_2^{\mathcal{H}(c_1,c_2,c_3)})^{\rho} \ \wedge\\
&\Big((C_2 = \tilde{h}_1^{\alpha_1}\tilde{h}_2^{\nu_4} \wedge C_1 = g_1^{\alpha_1}g_2^{\nu_5} \ \wedge\ C_3 = g_1^{\alpha_1^{e}}g_2^{\nu_6} \wedge\\
&\qquad g_1 = (C_3/g_1^{Pad_S(m)})^{\nu_7}g_2^{\nu_8} \ \wedge\ (1 \le \alpha_1 \le n-1)\Big) \vee\\
&\big(C_2 = (1/\tilde{h}_1)^{\alpha_2}\tilde{h}_2^{\nu_9} \wedge C_1 = g_1^{\alpha_2}g_2^{\nu_{10}} \ \wedge\ C_3 = g_1^{\alpha_2^{e}}g_2^{\nu_{11}} \wedge\\
&\qquad g_1 = (C_3/g_1^{Pad_S(m)})^{\nu_{12}}g_2^{\nu_{13}} \ \wedge\ (1 \le \alpha_2 \le n-1)\big) \vee\\
&\qquad \big(C_2 = \tilde{h}_1^{\alpha_3}\tilde{h}_2^{\nu_{14}} \ \wedge\ (n \le \alpha_3 \le p-n)\big)\Big)\Big)\Big)\Big\}.
\end{aligned}
$$

This protocol proves that either decryption fails or that both the encrypted value and p minus the encrypted value are either not in $[1, n-1]$ or not a valid RSA signature with respect to m, e, n, and Pad_S.

Selective conversion: If (c_1, c_2, c_3, c_4) is a valid confirmer signature, then the confirmer just return the decryption of (c_1, c_2, c_3, c_4) and otherwise answers $\perp$.

Remark 4. As the confirmation and the disavowal protocol involve double discrete logarithms, they are not very efficient because they use binary challenges. If batch verification technology [2] is incorporated, the computational load of both the verifier and the confirmer is about 20 times that of a similar proof with non-binary challenges. Furthermore, the protocol could be made more efficient by allowing non-binary challenges for parts of the protocol. Moreover, if e is small (e.g., 3 or 5), then there is a much more efficient way of proving the knowledge of a root of a discrete log (cf. [7]).

5 Alternative Generic Solutions

Although the two generic schemes presented in [21] are demonstrably insecure, they can both be modified such that they are provably secure in our model. In contrast to the scheme exhibited in Section 3, these schemes cannot provide perfect convertibility with respect to signature schemes such as RSA or DSS. However, they have instances where the confirmation and disavowal protocol is on order of magnitude more efficient than for the scheme described in the previous section. We note that the bi-proof proposed in [21] for disavowal is *not* computational zero-knowledge, however, it can be replaced by a similar proof that is perfect zero-knowledge (we refer to the full version of this paper for details).

The first scheme in [21] is based on signature schemes that are derived from 3-move honest-verifier zero-knowledge proofs of knowledge. The Schnorr signature scheme [26] is a typical example thereof. If an encryption scheme secure against adaptive chosen-ciphertext attacks is used as confirmer commitment scheme and the public keys of the signer and the confirmer are appended to the message that is signed, then the resulting confirmer signature scheme can be proven secure in our model provided that the underlying 3-move proofs of knowledge have the property that the third message is uniquely defined by the first two messages.

The second scheme in [21] is based on signature schemes that are existentially forgeable in their basic variant but become secure if a hash of the message is signed instead of the plain message. The RSA signature scheme is a typical representative for this class of signature schemes. Again, if an encryption scheme secure against adaptive chosen-ciphertext attacks is used, the public keys of the signer and the confirmer are appended to the message, and the signature scheme is deterministic, then the resulting confirmer signature scheme can be shown to be secure in our model.

Details will be given in the forthcoming full version of this paper.

6 Applications to Other Scenarios

As mentioned in [21], confirmer signatures schemes with conversion can be used to realize fair contract signing schemes as follows. The trusted third party in the contract signing scheme plays the role of the confirmer. Furthermore, recall that a signer can always confirm a valid confirmer signature. Thus, a confirmer signature scheme together with a confirmation protocol for the signer can be used to replace the "verifiable signature encryption scheme" in [1]: the parties issue confirmer signatures and prove the correctness of their respective signatures. After this step, either the real signatures can be exchanged or, if this fails, they can ask the TTP/confirmer to convert the confirmer signatures (a suitable policy for the TTP/confirmer should be included in the signed messages). The resulting optimistic fair contract signing scheme can be shown secure in the standard model (i.e., not in the random oracle model) if the security of the underlying signature scheme is assumed.

It is also possible to employ the techniques used for our confirmer signature scheme for realizing verifiable signature sharing schemes [18]. In a nutshell, a promise to a signature is split into shares according to a given secret sharing scheme. Then each of the shares is encrypted (similarly as the ordinary signature in our confirmer signature scheme) and it is proved that the encrypted values are indeed correct shares. Such a proof is similar as the confirmation protocol exhibited in Section 4. This approach is possible for signature schemes such as RSA or DSS. The resulting scheme will enjoy separability and be secure against adaptive attackers while previous solutions were either insecure [18] or secure only in a non-adaptive model [8, 19].

7 Acknowledgements

The authors are grateful to Victor Shoup for various discussions and to the anonymous referees for their helpful and detailed comments.

References

1. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *EUROCRYPT '98*, vol. 1403 of *LNCS*, pp. 591–606, 1998.
2. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT '98*, vol. 1403 of *LNCS*, pp. 236–250. Springer Verlag, 1998.
3. M. Bellare and S. Micali. How to sign given any trapdoor function. In *CRYPTO '88*, vol. 403 of *LNCS*, pp. 200–215. Springer-Verlag, 1990.
4. M. Bellare and P. Rogaway. The exact security of digital signature – how to sign with RSA and Rabin. In *EUROCRYPT '96*, vol. 1070 of *LNCS*, pp. 399–416. Springer Verlag, 1996.
5. F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT 2000*, vol. 1807 of *LNCS*, pp. 437–450. Springer Verlag, 2000 (this volume).
6. J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO '99*, vol. 1296 of *LNCS*, pp. 413–430, 1999.
7. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, vol. 1296 of *LNCS*, pp. 410–424. Springer Verlag, 1997.
8. D. Catalano and R. Gennaro. New efficient and secure protocols for verifiable signature sharing and other applications. In *CRYPTO '98*, vol. 1642 of *LNCS*, pp. 105–120. Springer Verlag, 1998.
9. D. Chaum. Designated confirmer signatures. In *EUROCRYPT '94*, vol. 950 of *LNCS*, pp. 86–91. Springer Verlag, 1994.
10. D. Chaum and H. van Antwerpen. Undeniable signatures. In *CRYPTO '89*, vol. 435 of *LNCS*, pp. 212–216. Springer-Verlag, 1990.
11. L. Chen. Efficient fair exchange with verifiable confirmation of signatures. In *ASIACRYPT '98*, vol. 1514 of *LNCS*, pp. 286–299. Springer Verlag, 1998.
12. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, vol. 839 of *LNCS*, pp. 174–187. Springer Verlag, 1994.
13. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO '98*, vol. 1642 of *LNCS*, pp. 13–25. Springer Verlag, 1998.
14. G. Di Crescenzo and R. Ostrovsky. On concurrent zero-knowledge with preprocessing. In *CRYPTO '99*, vol. 1296 of *LNCS*, pp. 485–502, 1999.
15. I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT 2000*, *LNCS*. Springer Verlag, 2000.
16. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.
17. C. Dwork and A. Sahai. Concurrrent zero-knowledge: Reducing the need for timing constraints. In *CRYPTO '98*, vol. 1642 of *LNCS*, pp. 105–120, 1998.
18. M. Franklin and M. Reiter. Verifiable signature sharing. In *EUROCRYPT '95*, vol. 921 of *LNCS*, pp. 50–63. Springer Verlag, 1995.

19. E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *EUROCRYPT '98*, vol. 1403 of *LNCS*, pp. 32–46. Springer Verlag, 1998.
20. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308.
21. M. Michels and M. Stadler. Generic constructions for secure and efficient confirmer signature schemes. In *EUROCRYPT '98*, vol. 1403 of *LNCS*, pp. 406–421, 1998.
22. K. Nguyen, Y. Mu, and V. Varadharajan. Undeniable confirmer signature. Proc. *Information Security Workshop'99*, *LNCS*, Springer-Verlag, 1999.
23. T. Okamoto. Designated confirmer signatures and public-key encryption are equivalent. In *CRYPTO '94*, vol. 839 of *LNCS*, pp. 61–74. Springer Verlag, 1994.
24. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, Feb. 1978.
25. J. Rompel. One-way functions are necessary and sufficient for secure signature. In *Proc. 22nd Annual ACM STOC*, 387–394, 1990.
26. C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.
27. M. Stadler. Publicly verifiable secret sharing. In *EUROCRYPT '96*, vol. 1070 of *LNCS*, pp. 191–199. Springer Verlag, 1996.

A An Insecure Confirmer Signature Scheme

This section show that the scheme due to Chen [11] is insecure because the confirmer can forge confirmer signatures of any message for an arbitrary signer.

Let us review this scheme briefly. Public system parameters are a group $G = \langle g \rangle$ and a prime $q = |G|$. The signer's public key is $y \in G$ and its secret key is $x = \log_g y$. The confirmer's public key is $z \in G$ and its secret key is $w = \log_g z$. Furthermore, a suitable hash function $\mathcal{H}$ is known. The signer generates a confirmer signature on m by picking $u, k_1, k_2 \in_R \mathbb{Z}_q$ and computing $\tilde{y} := y^u$, $\hat{y} := z^{xu}$, $r_1 := y^{k_1}$, $r_2 := z^{k_2}$, $c := \mathcal{H}(m, r_1, r_2)$, $s_1 := k_1 - uc \bmod q$, and $s_2 := k_2 - uxc \bmod q$. The resulting confirmer signature on m is given by $(c, s_1, s_2, \tilde{y}, \hat{y})$. It is valid if and only if $c = h(m, y^{s_1}\tilde{y}^c, z^{s_2}\hat{y}^c)$ and $\log_{\tilde{y}} \hat{y} = \log_g z$. We refer to [11] for a discussion of how the confirmer confirms/disavows.

This scheme is insecure because the confirmer can fake confirmer signatures for an arbitrary signer and message m: He picks random values $t, s_2, d \in_R \mathbb{Z}_q$ and computes $c := h(m, y^t, z^{s_2}y^d)$, $a := d/(wc) \mod q$, $\tilde{y} := y^a$, $\hat{y} := \tilde{y}^w$, and $s_1 := t - ac \mod q$. As $y^t = y^{s_1}\tilde{y}^c$ and $z^{s_2}y^d = z^{s_2}\hat{y}^c$ holds, $(c, s_1, s_2, \tilde{y}, \hat{y})$ is a confirmer signature on the message m. This attack is possible although the security of the scheme is proved in [11]. The problem is that it is erroneously assumed in the security proof that the knowledge extractor learns $log_z \hat{y}$, which is not necessarily the case.