A Practical Digital Multisignature Scheme
Based on Discrete Logarithms
(Extended Abstract)

Thomas Hardjono[1] * and Yuliang Zheng[2] **

[1] ATR Communications Research Laboratories
2-2 Hikaridai, Seika-Cho, Soraku-gun, Kyoto 619-02, Japan
[2] Department of Computer Science, University of Wollongong, Australia

Abstract. This paper proposes a practical digital multisignature scheme based on the $\mathcal{C}^{\star}_{sig}$ cryptosystem derived from the $\mathcal{C}_{sig}$ cryptosystem of Zheng and Seberry (1993). The simple scheme consists of three phases. In the first phase the issuer of the document prepares the document, the list of prospective signatories and a pad on which signatories are to write their signatures. In the second phase each signatory verifies the document, signs it and forwards it to the next signatory. In the third phase a trusted verifier or notary decides on the validity of the signatures. The scheme prevents cheating by dishonest signatories from going undetected. The scheme is practical and offers at least the same security level afforded by its underlying cryptosystem against external attacks. Internal attacks in the form of forging or cheating by a dishonest issuer or by one or more of the signatories (alone or by collaboration) requires the solving of instances of the discrete logarithm problem.

Keywords: Data Encryption, Information Security. Digital Signatures, Digital Multisignatures.

1 Introduction

The problem of providing digital signatures and digital multisignatures for electronic documents has been addressed by a number of researchers over the past years [1, 2, 3, 4, 5]. The aim of electronically signing an electronic document is in principle the same as handwritten signatures on paper documents. A useful analogy is that of the signing of a two-party contract. The party issuing the contract on paper requires the second party involved in the contract to sign it in the

---

* A substantial part of this work was completed when the author was at the Centre for Computer Security Research, Australia.
** Supported in part by the Australian Research Council under the reference number A49232172.

presence of a witness who is acceptable to both parties. The witness has the task of verifying that the signatures of both parties are authentic. This verification can take the form of a signature from the witness, which will be used to solve disputes that may occur in the future.

When this analogy is carried over into the world of computers a number of problems immediately occur. First and foremost is the problem of the signature representation to be attached to the document file. If signatures were to be represented electronically simply a unique string of bits to be appended to a document file, then both the document file and the signatures are vulnerable to modifications which may disadvantage the document issuer, the signatory or even the witness. Hence a method is required to ensure that any changes done to the signed document be detectable, the instant at which the validity of the document comes into question.

A number of sophisticated digital signature schemes have emerged in the past two decades. The strength of these schemes rely on their underlying cryptosystem which in turn must be resistant against various types of attacks. Chief among these attacks is the chosen ciphertext attack in which the attacker has access to the deciphering algorithm of a cryptosystem. In order to cryptanalyze the object ciphertext he or she can query the deciphering algorithm any number of times using any ciphertext (except the object ciphertext) as input to the deciphering algorithm. In this way he or she indirectly gains some knowledge about the object ciphertext that he or she wishes to cryptanalyze.

In this paper we present a practical digital multisignature scheme based on a modified version of the $\mathcal{C}_{sig}$ cryptosystem of [6]. The $\mathcal{C}_{sig}$ is one of a family of cryptosystems proposed in [6] which grew out of the earlier work in [7]. The family of cryptosystems represents an important step in public-key cryptography since they are practical and secure against *adaptively* chosen ciphertext attacks in a provable manner. In this paper we do not formally prove the security of the digital multisignature scheme since the constructs of the original $\mathcal{C}_{sig}$ are still intact. Hence the reader is directed to [6] for a formal proof of the security of the cryptosystem.

The outline of this paper is as follows. Section 2 presents the background notation for $\mathcal{C}_{sig}$. Section 3 presents the digital multisignature scheme, while Section 4 continues with a brief discussion on its security. This paper is closed with a conclusion in Section 5

2   Notation

The notation used in this paper is taken from the original work in [6], and is presented briefly in this section. The cryptosystems of [6] are reminiscent of the Diffie-Hellman cryptosystem [8] and El Gamal cryptosystem [9] in their use of a n-bit prime p (public) and a generator g (public) of the multiplicative group $GF(p)^*$ of the finite field $GF(p)$. Here n is a security parameter which is greater that 512 bits, while the prime p must be chosen such that $p - 1$ has a large prime factor [10]. In this paper the alphabet $\Sigma = \{0, 1\}$ will be employed, and

$|x|$ denotes length of a string x over Σ. Concatenation of string are denoted using the "$||$" symbol and the bit-wise XOR operations of two strings is symbolized using "$\oplus$". The notation $x_{[i...j]}$ $(i \leq j)$ is used to indicate the substring obtained by taking the bits of string x from the i-th bit (x_i) to the j-th bit (x_j). It is important to see that there is a natural one-to-one correspondence from strings in Σ^n to elements in the finite field $GF(2^n)$, and from from strings in Σ^n to integers in $[0, 2^n - 1]$.

Other notations are as follows. The action of choosing an element x randomly and uniformly from set S is denoted by $x \in_R S$. G is a cryptographically strong pseudo-random string generator based on the difficulty of computing discrete logarithms in finite fields [11, 12, 13]. G stretches an n-bit input string into an output string whose length can be an arbitrary polynomial in n. This generator produces $O(\log n)$ bits output at each exponentiation. The function h is a one-way hash function compressing input strings into n-bit output strings. All messages to be encrypted are chosen from the set $\Sigma^{P(n)}$, where $P(n)$ is an arbitrary polynomial with $P(n) \geq n$ and where padding can be used for messages of length less than n bits. The polynomial $\ell = \ell(n)$ specifies the length of tags. In the remainder of this paper all exponentiations are assumed to be done over the underlying groups. The reader is directed to [6] for a comprehensive discussion on the constructs of the family of cryptosystems.

The cryptosystem employed in this paper is a modified version of the $\mathcal{C}_{sig}$ cryptosystem of [6]. To distinguish it from $\mathcal{C}_{sig}$ we will refer to it as $\mathcal{C}_{sig}^\star$ The algorithm is shown in the following, where Bob having his secret-public key pair (x_B, y_B) wants to send a $P(n)$-bit message m to Alice whose key pair is (x_A, y_A). Here Alice's (Bob's) secret key x_A (x_B) is element chosen randomly from $[1, p-1]$ and $y_A = g^{x_A}$ $(y_B = g^{x_B})$. Bob enciphers using $E_{sig}^\star$ (Algorithm 1) while Alice deciphers using $D_{sig}^\star$ (Algorithm 2).

Algorithm 1 $E_{sig}^\star(x_B, y_A, p, g, m)$
    1. $k \in_R [1, p-1]$ s.t. $\gcd(k, p-1) = 1$.
    2. $r = y_A^{x_B + k}$.
    3. $z = G(r)_{[1...P(n)]}$.
    4. $c_1 = g^{x_B}$.
    5. $c_2 = g^k$.
    6. $t = (h(m) - x_B r)/k \bmod (p-1)$.
    7. $c_3 = c_2^t$.
    8. $c_4 = z \oplus m$.
    9. output (c_1, c_2, c_3, c_4).
end

Algorithm 2 $D^\star_{sig}(x_A, p, g, c_1, c_2, c_3, c_4)$

    1. $r' = (c_1 c_2)^{x_A}$.

    2. $z' = G(r')_{[1 \cdots P(n)]}$.

    3. $m' = z' \oplus c_4$.

    4. if $g^{h(m')} = c_1^{r'} c_3$ then

        output (m')

    else

        output $(\emptyset)$.

end

To distinguish between the problem of message tampering during transit and the problem of a dishonest issuer or signatory we will employ the terms *integrity* and *authenticity*. Corresponding to this is the difference between a *message* and a *document*, the later being contained in the former. A receiver of a message must himself of herself verify the integrity of the message, namely the fact that the document and signatures so far accumulated in the message has arrived from the sender unmodified. He or she must also verify the authenticity of the document, namely the fact that it is the same as the original document produced by the issuer of the document. These two terms will become clear in the next section, in which authenticity refers to the state of the document as produced by the issuer and integrity refers to the safety of the message containing the document with the signatures so far collected during their transit from sender to a receiver.

3   A Digital Multisignature Scheme

The scheme based on the $\mathcal{C}^\star_{sig}$ cryptosystem follows the traditional approach of digital signatures where an *issuer* S_I of a *document* D requires a number of *signatories* $S_1, S_2, \ldots, S_u$ to electronically sign the document, which is to be verified by a *trusted verifier* S_V at the end of the signing process. More specifically, the issuer requires the signature of u number of signatories, each assumed to hold a secret key x_i and a public key $y_i = g^{x_i}$ ($i = 1, \ldots, u$). The issuer's secret and public key are similarly denoted as x_I and y_I respectively. In addition x_V and y_V are the secret and public keys of the verifier.

The digital multisignature scheme consists of three phases. In the first phase the issuer prepares the document and the "list" of the identities of the prospective signatories of the document. The issuer then submits the document-list pair to the verifier who selects randomly and uniformly a value x_d (secret) associated with the document-list. The verifier also calculates $y_d = g^{x_d}$ and broadcasts y_d. The value x_d can be looked upon as a unique document identifier, and can in fact be derived from the secret random number and the true document or contract number. The tag associated with document-list mirrors the case where a document issuer must choose a trusted witness or notary that is allowed to view the document before it is passed around all the signatories who in turn require such an action be taken by the issuer before they enter into the signing process.

On receiving a tag for the document-list from the verifier the issuer creates an authenticator incorporating the document-list and the tag. The issuer then sends a message to the first signatory containing the document, pad and other parameters necessary for the authentication process by the signatory.

In the second phase the first signatory S_1 tries to establish the authenticity of the document-list pair and the integrity of the message that carried the document-list and pad. If the document has not been tampered with during its transit and the document is authentic (not substituted illegally by the issuer) then the signatory S_1 signs the pad and forwards the document-list pair and pad to signatory S_2. This verification, signing and forwarding process is repeated by each subsequent signatory S_i $(i = 2, \ldots, u)$, with the last signatory S_u forwarding the document-list and pad to the verifier. The second phase is then completed.

In the third phase the trusted verifier receives the message and tries to establish its integrity. The verifier then checks the authenticity of the document-list and checks whether all signatories have signed. Before starting the verification process the verifier broadcasts x_d. Using this value all signatories and the general public can check whether $g^{x_d} = y_d$.

In the following the three phases will be presented more precisely. The document D is assumed to contain a list of prospective signatories and the verifier, similar to the way that multi-party agreements on paper documents have a list of the names of signatories and a witness to the signing of the document. Throughout this paper it is assumed that an electronic document contains a unique identifier and a timestamp for duration of time in which the signing process must be completed. This is to prevent replay of messages by an active attacker.

Phase 1 In the first phase the issuer prepares the document D and the list PAD, and registers to the verifier $M = (D \parallel PAD)$ through $V_{register}$ (Algorithm 3). The verifier then chooses x_d, broadcasts $y_d = g^{x_d}$ and returns the tag $g^{h(M \oplus x_d)}$ for M. The issuer then signs the pad and sends $(C_{I_1}, C_{I_2}, C_{I_3}, c_1)$ to the first signatory S_1. The actions of the issuer is shown in E_I (Algorithm 4).

> **Algorithm 3** $V_{register}(M, p, g)$
>   1. $x_d \in_R [1, p-1]$ such that $\gcd(k_I, p-1) = 1$.
>   2. $y_d = g^{x_d}$.
>   3. $\alpha = h(M \oplus x_d)$.
>   4. $\beta = g^{\alpha}$.
>   5. broadcast y_d.
>   6. output (β).
> **end**

Algorithm 4 $E_I(x_I, y_1, \ldots, y_u, y_V, p, g, D)$

    1. $k_I \in_R [1, p-1]$ such that $\gcd(k_I, p-1) = 1$.

    2. $C_{I_1} = g^{x_I}$.

    3. $C_{I_2} = g^{k_I}$.

    4. $PAD = \left(y_I^{h(D)}\right)^{x_I+k_I} \oplus \left(y_1^{h(D)}\right)^{x_I+k_I} \oplus \cdots \oplus \left(y_u^{h(D)}\right)^{x_I+k_I} \oplus \left(y_V^{h(D)}\right)^{x_I+k_I}$.

    5. $M = D \parallel PAD$.

    6. $R = V_{register}(M)$.

    7. $A = (h(MR) - x_I y_d)/k_I \mod (p-1)$.

    8. $T = (C_{I_2})^A$.

    9. $pad_1 = ((C_{I_1} C_{I_2})^{x_I})^{h(D)}$.

    10. $m_1 = M \parallel R \parallel T \parallel pad_1$.

    11. $r_1 = y_1^{x_I+k_I}$.

    12. $z_1 = G(r_1)_{[1 \ldots P(n)]}$.

    13. $t_1 = (h(m_1) - x_I r_1)/k_I \mod (p-1)$.

    14. $C_{I_3} = (C_{I_2})^{t_1}$.

    15. $c_1 = z_1 \oplus m_1$.

    16. output $(C_{I_1}, C_{I_2}, C_{I_3}, c_1)$.

end

The important components of the message of m_1 are T and pad_1. T ensures the authenticity of M and thus D, while pad_1 represents the signature of the issuer S_I who is also involved in the event. All three M, R and T remain unchanged throughout the remainder of the signing process. Each signatory and the verifier uses T to verify that no malicious signatory has modified M. In contrast, t_1 is used only by the next signatory to verify the integrity of m_1 containing the signed pad. Note that Steps 11 to 16 is the basic encryption steps of the $\mathcal{C}^{\star}_{sig}$ cryptosystem in order to provide the secure and authentic transfer of m_1.

Phase 2 The second phase is started when the first signatory S_1 verifies the integrity and authenticity of the unsigned document using D_{S_1} (Algorithm 5). If all is well the signatory S_1 keeps a copy of the values $(C_{I_1}, C_{I_2}, C_{I_3}, c_1)$ which he or she received and verified.

Algorithm 5 $D_{S_1}(x_1, y_d, p, g, C_{I_1}, C_{I_2}, C_{I_3}, c_1)$

    1. $r'_1 = (C_{I_1} C_{I_2})^{x_1}$.

    2. $z'_1 = G(r'_1)_{[1...P(n)]}$.

    3. $m'_1 = z'_1 \oplus c_1$.

    4. if $g^{h(m'_1)} = (C_{I_1}{}^{r'_1}) C_{I_3}$ then

        output (m'_1) and continue to Step 5

      else

        output $(\emptyset)$ and stop.

    5. Separate m'_1 into M, R, T and pad_1.

    6. if $g^{h(MR)} = (C_{I_1}{}^{y_d}) T$ then

        output (M)

      else

        output $(\emptyset)$ and stop.

end

Next the signatory S_1 signs the pad using E_{S_1} (Algorithm 6) and forwards the message containing the document and pad to signatory S_2. Signatory S_1 also keeps a copy of the message $(C_{I_1}, C_{I_2}, C_{I_3}, c_{2_1}, c_{2_2}, c_{2_3}, c_{2_4})$ that he or she sent to S_2. Note that the output of E_{S_1} has more components compared to the output of E_I. This has no effect on the scheme as a whole since the components $(C_{I_1}, C_{I_2}, C_{I_3})$ important for the verification of the authenticity of the document is the same as in E_I. Each subsequent signatories after S_1 will receive the same format of the message as that output by E_{S_1}.

Algorithm 6 $E_{S_1}(x_1, y_2, p, g, C_{I_1}, C_{I_2}, C_{I_3}, D, M, R, T, pad_1)$

    1. $k_1 \in_R [1, p-1]$ such that $\gcd(k_1, p-1) = 1$.

    2. $c_{2_1} = g^{x_1}$.

    3. $c_{2_2} = g^{k_1}$.

    4. $pad_2 = pad_1 \oplus ((C_{I_1} C_{I_2})^{x_1})^{h(D)}$.

    5. $m_2 = M \parallel R \parallel T \parallel pad_2$.

    6. $r_2 = y_2^{x_1 + k_1}$.

    7. $z_2 = G(r_2)_{[1...P(n)]}$.

    8. $t_2 = (h(m_2) - x_1 r_2)/k_1 \bmod (p-1)$.

    9. $c_{2_3} = (c_{2_2})^{t_2}$.

    10. $c_{2_4} = z_2 \oplus m_2$.

    11. output $(C_{I_1}, C_{I_2}, C_{I_3}, c_{2_1}, c_{2_2}, c_{2_3}, c_{2_4})$.

end

In general the signatory S_i $(1 \leqq i \leqq u)$ employs E_{S_i} (Algorithm 7) to sign the pad associated with the document, and to prepare the document and the pad for the next signatory S_{i+1}. Note that for E_{S_i} signatory S_0 is identical to the issuer S_I and that signatory S_{u+1} is equivalent the verifier.

Algorithm 7 $E_{S_i}(x_i, y_{i+1}, p, g, C_{I_1}, C_{I_2}, C_{I_3}, D, M, R, T, pad_i)$

    1. $k_i \in_R [1, p-1]$ such that $\gcd(k_i, p-1) = 1$.
    2. $c_{(i+1)_1} = g^{x_i}$.
    3. $c_{(i+1)_2} = g^{k_i}$.
    4. $pad_{i+1} = pad_i \oplus ((C_{I_1} C_{I_2})^{x_i})^{h(D)}$.
    5. $m_{i+1} = M \parallel R \parallel T \parallel pad_{i+1}$.
    6. $r_{i+1} = y_{i+1}^{x_i + k_i}$.
    7. $z_{i+1} = G(r_{i+1})_{[1 \ldots P(n)]}$.
    8. $t_{i+1} = (h(m_{i+1}) - x_i r_{i+1})/k_i \bmod (p-1)$.
    9. $c_{(i+1)_3} = (c_{(i+1)_2})^{t_{i+1}}$.
    10. $c_{(i+1)_4} = z_{i+1} \oplus m_{i+1}$.
    11. output $(C_{I_1}, C_{I_2}, C_{I_3}, c_{(i+1)_1}, c_{(i+1)_2}, c_{(i+1)_3}, c_{(i+1)_4})$.
end

The signatory S_i uses D_{S_i} (Algorithm 8) to verify the integrity and authenticity of the document and pad from signatory S_{i-1}. Phase 2 is ended when the last signatory S_u employs D_{S_i} to verify the message he or she received from signatory S_{u-1} and employs E_{S_i} to sign the document and prepare the enclosing message for the verifier. Note that each signatory S_i must keep a copy of the message $(C_{I_1}, C_{I_2}, C_{I_3}, c_{i_1}, c_{i_2}, c_{i_3}, c_{i_4})$ received from signatory S_{i-1} and a copy of the message $(C_{I_1}, C_{I_2}, C_{I_3}, c_{(i+1)_1}, c_{(i+1)_2}, c_{(i+1)_3}, c_{(i+1)_4})$ sent to signatory S_{i+1}.

Algorithm 8 $D_{S_i}(x_i, y_d, p, g, C_{I_1}, C_{I_2}, C_{I_3}, c_{i_1}, c_{i_2}, c_{i_3}, c_{i_4})$

    1. $r_i' = (c_{i_1} c_{i_2})^{x_i}$.
    2. $z_i' = G(r_i')_{[1 \ldots P(n)]}$.
    3. $m_i' = z_i' \oplus c_{i_4}$.
    4. if $g^{h(m_i')} = (c_{i_1}^{r_i'}) c_{i_3}$ then
        output (m_i') and continue to Step 5
      else
        output $(\emptyset)$ and stop.
    5. Separate m_i' into M, R, T and pad_i.
    6. if $g^{h(MR)} = (C_{I_1}^{y_d}) T$ then
        output (M)
      else
        output $(\emptyset)$ and stop.
end

Phase 3 In Phase 3 the verifier receives the document and pad from the last signatory S_u and verifies their authenticity and integrity. These tasks are done using D_V (Algorithm 9) which is similar in form to D_{S_i}. However, D_V contains the additional steps of checking that the $pad_{u+1} = pad_V$ received from signatory S_u contains the signature of all signatories S_i ($i = 1, \ldots, u$). Furthermore, after broadcasting x_d, which is used by all participants to check y_d, the verifier must see that M is still authentic using x_d. Each signatory can in fact check whether $R = g^{h(M \oplus x_d)}$.

Algorithm 9 $D_V(x_V, x_d, y_d, p, g, C_{I_1}, C_{I_2}, C_{I_3}, c_{V_1}, c_{V_2}, c_{V_3}, c_{V_4})$

1. $r'_V = (c_{V_1} c_{V_2})^{x_V}$.
2. $z'_V = G(r'_V)_{[1 \ldots P(n)]}$.
3. $m'_V = z'_V \oplus c_{V_3}$.
4. if $g^{h(m'_V)} = (c_{V_1}{}^{r'_V}) c_{V_3}$ then
      output (m'_V) and continue to Step 5
   else
      output $(\emptyset)$ and stop.
5. Separate m'_V into M, R, T and pad_V.
6. if $g^{h(MR)} = (C_{I_1}{}^{y_d}) T$ then
      output (M) and continue to Step 7
   else
      output $(\emptyset)$ and stop.
7. Separate M into D and PAD.
8. $pad_{check} = pad_V \oplus ((C_{I_1} C_{I_2})^{x_V})^{h(D)}$.
9. if $pad_{check} = PAD$
      output message "All Signed"
   else
      output $(\emptyset)$ and stop.

end

4   Security

The third cryptosystem of [6] (instead of the first or second cryptosystems) was deemed suitable for the foundation of the multisignature scheme since it already featured a basic authentication capability in the sense of El Gamal [9]. In [6] it was shown that both the first and the second cryptosystems are semantically secure against adaptively chosen ciphertext attacks under the assumption that the space induced by the two cryptosystems were sole-samplable (see [6]). However, $\mathcal{C}_{sig}$ was not able to be proven as being semantically secure against chosen plaintext attacks due to the difficulty in measuring the leak of information m from $t = (h(m) - xr)/k \bmod (p-1)$ in the ciphertext. Thus $\mathcal{C}_{sig}$ is semantically secure against adaptively chosen ciphertext attacks only with the assumption that it is semantically secure against chosen plaintext attacks.

Our proposed $\mathcal{C}^{\star}_{sig}$ algorithm improves $\mathcal{C}_{sig}$ by reducing this possible leak. Previously a receiver of a message would already know the values of m and r at the successful decipherment of the ciphertext message. Using these two values he or she could work towards obtaining x and k from $t = (h(m) - xr)/k \bmod (p-1)$. The $\mathcal{C}^{\star}_{sig}$ makes harder this effort of obtaining x and k from t at the expense of an apparent ease in substituting m by using the known r, $c_1 (= g^x)$ and $c_2 (= g^k)$. Note, however, that in our application of the $\mathcal{C}^{\star}_{sig}$ algorithm the registration value $R = g^{h(M \oplus x_d)}$ guards against this trivial substitution. In the case of the typical one-to-one communication between two trusting parties $\mathcal{C}^{\star}_{sig}$ represents a more secure approach against external attacks.

From the point of view of a dishonest signatory (excluding the issuer and the verifier) the security of the digital multisignature scheme lies in the strength of

T (Algorithm 4). Unlike the case of the native form of $\mathcal{C}_{sig}$ and $\mathcal{C}_{sig}^{\star}$ where the attacker's aim is to derive and modify M, in this case each signatory already has M on receipt and correct decipherment of the message. Thus, his or her aim would be to cheat the remainder of signatories into signing a false document D' (note that a cheating signatory cannot do more than some random modifications to the PAD). In order to do this he or she must forge T, which requires knowing x_I, k_I and x_d. Deriving x_I from C_{I_1}, k_I from C_{I_2} and x_d from y_d at the very least constitutes the solving of instances of the discrete logarithm problem.

One possible scenario is when the first and the last signatories (S_1 and S_u) collaborate to cheat the rest of the signatories. This can be done if they can obtain x_I and k_I. The first signatory then calculates a false D' and makes M'. He or she then calculates A' using a false x_I' and k_I', and in turn generates T' using A' (Algorithm 4). Hence signatories $S_2, \ldots, S_{u-1}$ will be signing a forged document M', while the last signatory can restore T before signing the pad and delivering them to the verifier. Note, however, that if D is modified into D', signatory S_i ($2 \leq i \leq u-1$) will be signing using $((C_{I_1} C_{I_2})^{x_i})^{h(D')}$. Thus the test of $pad_{check} = PAD$ by the verifier will fail at the first instant (Algorithm 9). The value PAD cannot be intelligently modified by anyone (except the issuer) since breaking it at least constitutes the solving discrete logarithm problems. Random modifications of PAD will also result in the failure of the $pad_{check} = PAD$ test by the verifier. Each signatory will be able to verify the M on which they based their signatures in the third phase when x_d is broadcasted. If the issuer cheats (alone or by collaboration) by creating $M' = D' \parallel PAD'$, then it will be detected by the verifier when he or she computes $R' = g^{h(M' \oplus x_d)}$ in the third phase.

Though unlikely to occur in real world situations where all signatories to a contract are aware of the contract, one possible weakness of the scheme is that a dishonest issuer can discredit some parties by mentioning their identities in D, yet purposely fail to include them in the computation of PAD. The issuer can get away with this only by the collaboration of all actual signatories $S_1, \ldots, S_u$. For if even one of these signatories is honest, he or she may deliver the document to be signed to one or more of the discredited parties. They who would then unknowingly add their signatures to the pad, thus frustrating the efforts of the dishonest issuer. This problem can be avoided by allowing the verifier to generate the PAD within $V_{register}$ using x_d and a random k_d. However, a better solution to this problem remains for future investigation, where the verifier should be able to know precisely the identities of those who have actually signed the pad.

5   Conclusion

In this paper we have presented a practical digital multisignature scheme based on the $\mathcal{C}_{sig}^{\star}$ cryptosystem derived from the $\mathcal{C}_{sig}$ cryptosystem of [6]. The scheme, which consists of three phases, allows for the detection of a dishonest issuer of the document and of dishonest signatories. Any dishonesty by a signatory is detected in the first place by the next signatory during the second phase, and in the second place by the verifier and all signatories in the third phase. Any

detected dishonesty will result in the verifier annulling any effects of the contract specified in the document. The scheme is practical and offers at the very least the same security level afforded by the $\mathcal{C}_{sig}$ cryptosystem of [6] against external attacks. Internal attacks in the form of forging or cheating by a dishonest issuer or by one or more of the signatories (alone or by collaboration) requires the solving of instances of the discrete logarithm problem.

Acknowledgements

References

1. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–128, 1978.
2. D. W. Davies, "Applying the RSA digital signature to electronic mail," *IEEE Computer*, vol. 16, pp. 55–62, February 1983.
3. A. Fiat and A. Shamir, "How to prove yourself: practical solutions of identification and signature problems," in *Advances in Cryptology - Proceedings of Crypto'86*, Lecture Notes in Computer Science, Vol. 263, pp. 186–194, Springer-Verlag, 1987.
4. T. Okamoto, "A digital multisignature scheme using bijective public-key cryptosystems," *ACM Transactions on Computer Systems*, vol. 6, no. 8, pp. 432–441, 1988.
5. K. Ohta and T. Okamoto, "A digital multi-signature scheme based on the fiat-shamir scheme," in *Advances in Cryptology - Proceedings of ASIACRYPT '91*, (Fujiyoshida, Japan), pp. 75–79, November 1991.
6. Y. Zheng and J. Seberry, "Immunizing public key cryptosystems against chosen ciphertext attacks," *IEEE Journal of Selected Areas in Communications*, 1993. (to appear).
7. Y. Zheng, T. Hardjono, and J. Seberry, "A practical non-malleable public key cryptosystem," Technical Report CS91/28, Computer Science Department, University College, University of New South Wales, Australia, 1991.
8. W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.
9. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, pp. 469–472, 1985.
10. B. A. LaMacchia and A. M. Odlyzko, "Computation of discrete logarithms in prime fields," *Designs, Codes and Cryptography*, vol. 1, pp. 47–62, 1991.
11. M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984.
12. D. L. Long and A. Wigderson, "The discrete logarithm hides O($\log n$) bits," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 363–372, 1988.
13. R. Peralta, "Simultaneous security of bits in the discrete log," in *Advances in Cryptology - Proceedings of EuroCrypt'85*, Lecture Notes in Computer Science, Vol. 219 (F. Pichler, ed.), pp. 62–72, Springer-Verlag, 1985.

This article was processed using the L^AT_EX macro package with LLNCS style