Publicly Verifiable Secret Sharing

Markus Stadler *

Institute for Theoretical Computer Science
ETH Zurich
CH-8092 Zurich, Switzerland
Email: `stadler@inf.ethz.ch`

Abstract. A secret sharing scheme allows to share a secret among several participants such that only certain groups of them can recover it. Verifiable secret sharing has been proposed to achieve security against cheating participants. Its first realization had the special property that everybody, not only the participants, can verify that the shares are correctly distributed. We will call such schemes publicly verifiable secret sharing schemes, we discuss new applications to escrow cryptosystems and to payment systems with revocable anonymity, and we present two new realizations based on ElGamal's cryptosystem.

1 Introduction

A *secret sharing* scheme [20, 2] allows to split a secret into different pieces, called shares, which are given to the participants, such that only certain groups of them can recover the secret. The first secret sharing schemes have been threshold schemes, where only groups of more than a certain number of participants can recover the secret.

Verifiable secret sharing (VSS) is a cryptographic primitive proposed in [7] to achieve security against cheating participants. A verification protocol allows the honest participants to ensure that they can recover a unique secret. VSS plays an important role in the design of protocols for secure multi-party computation (see e.g. [1]). The first realization of VSS, presented in [7], has the very special property that not only the participants, but everybody is able to verify that the shares have been correctly distributed. We will call such schemes *publicly verifiable secret sharing (PVSS)* schemes. Apart from the applications for "ordinary" VSS, PVSS can be used for new escrow-cryptosystems, and for the realization of digital payment systems with revocable anonymity.

The main technical results of this paper are two new PVSS schemes which can also be used with general (monotone) access structures. Both schemes are based on ElGamal's cryptosystem [9]. Furthermore, the security of the first scheme can be proved to be equivalent to some well-known cryptographic problems.

* Supported by the Swiss Federal Commission for the Advancement of Scientific Research (KWF) and by the Union Bank of Switzerland.

2 Publicly Verifiable Secret Sharing and Applications

2.1 An Informal Model of PVSS

In the sequel we give a informal description of secret sharing, verifiable secret sharing, and publicly verifiable secret sharing. It is not our goal to present a precise mathematical definition, but to illustrate the basic properties of the schemes and to point out the difference between ordinary and publicly verifiable secret sharing.

A secret sharing scheme consists of a *dealer*, n participants $P_1, \ldots, P_n$, and an access structure $\mathcal{A} \subseteq 2^{\{1,\ldots,n\}}$. The access structure is monotone, which means that if $A \in \mathcal{A}$ and $A \subseteq B$ then $B \in \mathcal{A}$. For instance, in a threshold secret sharing scheme with threshold k the access structure is defined as $\mathcal{A} = \{A \in 2^{\{1,\ldots,n\}} \mid |A| \geq k\}$, which means that any coalition of at least k participants can recover the secret.

To share a secret s among the participants, the dealer runs an algorithm *Share*

$$\mathit{Share}(s) = (s_1, \ldots, s_n)$$

to compute the shares. The dealer then sends each share s_i secretly to P_i, $i = 1, \ldots, n$. If a group of participants wants to recover the secret, they run an algorithm *Recover*, which has the property that

$$\forall A \in \mathcal{A}: \quad \mathit{Recover}(\{s_i | i \in A\}) = s \ ,$$

and that for all $A \notin \mathcal{A}$ it is computationally infeasible to calculate s from $\{s_i | i \in A\}$. Thus, only those coalitions of participants defined by the access structure $\mathcal{A}$ are able to recover the secret s. A secret sharing scheme is called *perfect* if for all $A \notin \mathcal{A}$ the shares $\{s_i | i \in A\}$ give no Shannon information about the secret.

One problem of such secret sharing schemes is that they are not secure against cheating participants who send false shares when the secret is to be recovered. Another problem is that a cheating dealer could distribute false shares, so that different groups of participants recover different secrets. Such problems arise in protocols for secure multi-party computations (see e.g. [1]), and can be solved with verifiable secret sharing (VSS) schemes [7].

A VSS scheme is a secret sharing scheme with an additional, possibly interactive algorithm *Verify* which allows the participants to verify the validity of their shares:

$$\exists u \ \forall A \in \mathcal{A}: \quad (\forall i \in A: \mathit{Verify}(s_i) = 1) \quad \Rightarrow \quad \mathit{Recover}(\{s_i | i \in A\}) = u,$$
$$\textit{and } u = s \textit{ if the dealer was honest.}$$

In other words, all groups of participants recover the same value if their shares are valid, and this unique value is the secret if the dealer was honest. A VSS scheme is called non-interactive, if the algorithm *Verify* requires no interaction between the participants [11].

But even with a non-interactive VSS scheme, the participants can verify the validity of only their own shares, but they cannot know whether other participants (with whom they might be able to recover the secret) have also received valid shares. This problem can be solved with publicly verifiable secret sharing (PVSS). In a PVSS scheme a public encryption function E_i is assigned to each participant P_i, such that only he knows the corresponding decryption function D_i. The dealer now uses the public encryption functions to distribute the shares by calculating

$$S_i = E_i(s_i), \quad i = 1, \ldots, n$$

and publishing the encrypted shares S_i. To verify the validity of all the encrypted shares, there is an algorithm *PubVerify* with the property that

$$\begin{aligned} &\exists u \ \forall A \in 2^{\{1,\ldots,n\}} : \\ &\qquad (\mathit{PubVerify}(\{S_i | i \in A\}) = 1) \ \Rightarrow \ \mathit{Recover}(\{D_i(S_i) | i \in A\}) = u \end{aligned}$$

and $u = s$ if the dealer was honest. In other words: If a set of encrypted shares is "good" according to *PubVerify*, then the honest participants can decrypt them and recover the secret. Note that *PubVerify* can be executed even if the participants have not received their shares so far. To run *PubVerify* it may be necessary to communicate with the dealer (but not with any participant). A PVSS scheme is called non-interactive if *PubVerify* requires no interaction with the dealer at all.

Theoretically, PVSS could be realized using techniques of [4] to prove (or to argue about) the satisfiability of a circuit, but this would be very inefficient. We will present more practical solutions for sharing discrete logarithms and for sharing *e-th* roots in Sections 3 and 4, respectively. Both schemes are based on ElGamal's cryptosystem [9].

2.2 Applications of PVSS

Apart from the applications of ordinary secret sharing and of VSS, there are two interesting problems for which PVSS can be used. One of those is software key escrow, such as Micali's fair cryptosystems [16]. The basic idea of fair cryptosystems is that any user shares his secret key among several trustworthy (from the user's point of view) escrow agents by means of a VSS scheme. Each escrow agent can verify that he obtained a correct share of the secret key. However, one problem of such fair cryptosystems is that the recipient of an encrypted message decides on the set of trustworthy escrow agent, although the sender of the message might trusts a different set of agents. Furthermore, the set of escrow agents can only be changed by changing the key. A better solution for this problem would be to have the sender provide information for the escrow agents to decrypt the message. Such a system could be realized using a non-interactive PVSS scheme with an access structure that allows the recipient as well as a the escrow agents to recover (i.e. decrypt) the message. Since the encrypted shares can be publicly verified, everybody, e.g. any network provider, can verify that the message could be recovered by a legitimate subset of escrow agents.

Another application of PVSS is the design of electronic cash systems providing revocable anonymity [5, 21, 15, 6]. Payments made with such systems are (usually) not traceable to the payer, but if the anonymity of the scheme is abused for criminal activities, the payer's identity can be recovered with the help of so-called trustees or judges. PVSS could be used to verifiably encrypt tracing-information for the trustees in a transaction without compromising the anonymity of that transaction.

3 PVSS for Sharing Discrete Logarithms

We will first describe two well-known methods for verifiably sharing discrete logarithms. The verification for both schemes consists of checking whether the secret share is the discrete logarithm of a publicly known element. Therefore, these schemes can be extended to PVSS schemes by means of an encryption scheme that allows to verify that a cipher-text contains the discrete logarithm of a given value. Let us first briefly describe the number-theoretical setting.

3.1 Double Exponentiation and Double Discrete Logarithms

Let p be a large prime so that $q = (p-1)/2$ is also prime[1], and let $h \in \mathbb{Z}_p^*$ be an element of order q. Let further G be a group of order p, and let g be a generator of G so that computing discrete logarithms to the base g is difficult.

Our scheme will make use of double exponentiation. By double exponentiation with bases g and h we mean the function

$$\mathbb{Z}_q \rightarrow G: \; x \mapsto g^{(h^x)} \; .$$

By the double discrete logarithm of $y \in G$ to the bases g and h we mean the unique $x \in \mathbb{Z}_q$ with

$$y = g^{(h^x)}$$

if such an x exists.

3.2 Verifiable Sharing of Discrete Logarithms

Let $s \in \mathbb{Z}_p$ be the secret value and let $S = g^s$ be publicly known. There are different ways to verifiably share this secret. We first present a solution for general monotone access structures, and then briefly describe a threshold scheme [11, 18].

Let $\mathcal{A}$ be a monotone access structure. For each $A = \{j_1, \ldots, j_k\} \in \mathcal{A}$, the dealer proceeds as follows: He computes the secret shares

$$s_{Ai} = \begin{cases} \text{randomly chosen in } \mathbb{Z}_p \text{ for } i = j_1, \ldots, j_{k-1} \\ s - \sum_{\ell=1}^{k-1} s_{Aj_\ell} \pmod{p} \text{ for } i = j_k \end{cases}$$

[1] This property is necessary in order to prove the security of the scheme.

and secretly sends s_{Ai} to the participant P_i. The values $S_{Ai} = g^{s_{Ai}}$ are published so that everybody can verify that

$$\forall A \in \mathcal{A} : \prod_{i \in A} S_{Ai} = S.$$

The participant P_i can verify his share by checking whether s_{Ai} is the discrete logarithm of S_{Ai}. Note that this construction is quite unpractical for large access-structures.

To share s in a threshold-scheme with threshold k, a publicly-known element $x_i \in \mathbb{Z}_p$, $x_i \neq 0$ is assigned to each participant P_i. The dealer chooses random elements $f_j \in \mathbb{Z}_p$, $j = 1, \ldots, k-1$, and publishes the values $S = g^s$ and $F_j = g^{f_j}$, $j = 1, \ldots, k-1$. Then he secretly sends to each P_i the share

$$s_i = s + \sum_{j=1}^{k-1} f_j x_i^j \pmod{p}$$

Any group of at least k participants can now compute s using Lagrange's interpolation formula. To verify a share s_i, the participant P_i can compute

$$S_i = S \cdot \prod_{j=1}^{k-1} F_j{}^{(x_i^j)}$$

and check whether $S_i = g^{s_i}$. See [11, 18] for further details.

To make these schemes publicly verifiable, we need a public-key encryption scheme that allows to verifiably encrypt the discrete logarithm of a publicly known element. In other words, given a cipher-text W and a group element S, it should be possible to convince everybody that the recipient obtains $\log_g S$ by decrypting W.

3.3 Verifiable Encryption of Discrete Logarithms

Our encryption scheme is identical to ElGamal's public key system [9], which is a variation of the Diffie-Hellman key-exchange protocol [8].

First, each participant randomly chooses a secret key $z \in \mathbb{Z}_q$ and publishes his public-key $y = h^z \pmod{p}$. To encrypt a message $m \in \mathbb{Z}_p^*$ with the public-key y, the dealer randomly chooses $\alpha \in \mathbb{Z}_q$ and calculates the pair

$$(h^\alpha, m^{-1} \cdot y^\alpha) \pmod{p}.$$

The cipher-text (A, B) can be decrypted by the recipient by calculating

$$m = A^z / B \pmod{p}.$$

Let us now describe a protocol for verifying that a pair (A, B) encrypts the discrete logarithm of a public element $V = g^v$ of the group G. It is based on the fact that if (A, B) is equal to $(h^\alpha, v^{-1} \cdot y^\alpha) \pmod{p}$ for any $\alpha \in \mathbb{Z}_q$ then

$$V^B = g^{vB} = g^{(y^\alpha)}.$$

The prover (who will be the dealer in the secret sharing scheme) now proves to the verifier that the discrete logarithm of A to the base h is identical to the double discrete logarithm of V^B to the bases g and y.

Prover		**Verifier**
repeat K times:		
$w \in_R \mathbb{Z}_q$ $t_h = h^w \pmod{p}$ $t_g = g^{(y^w)}$		
	$\xrightarrow{t_h,\, t_g}$	
		$c \in_R \{0,1\}$
	$\xleftarrow{c}$	
$r = w - c \cdot \alpha \pmod{q}$		
	$\xrightarrow{r}$	
		$t_h \overset{?}{=} h^r A^c \pmod{p}$
		$t_g \overset{?}{=} \begin{cases} g^{(y^r)} & \text{if } c = 0 \\ V^{(B \cdot y^r)} & \text{if } c = 1 \end{cases}$

With the techniques of [12] for converting an identification scheme into a signature scheme, combined with ideas from [19], we can construct a non-interactive "proof": Let $\mathcal{H}_\ell : \{0,1\}^* \to \{0,1\}^\ell$ be a cryptographically strong hash-function ($\ell \approx 100$). For $i = 1 \ldots \ell$, the prover chooses $w_i \in_R \mathbb{Z}_q$ and calculates $t_{hi} = h^{w_i} \pmod{p}$, and $t_{gi} = g^{(y^{w_i})}$. Then he computes the ℓ-tuple

$$R = (r_1, \ldots, r_\ell) = (w_1 - c_1\alpha \pmod{q}, \ldots, w_\ell - c_\ell\alpha \pmod{q})$$

where c_i denotes the i-th bit of

$$c = \mathcal{H}_\ell(V||A||B||t_{h1}||t_{g1}||\ldots||t_{h\ell}||t_{g\ell}) \qquad (*)$$

The non-interactive proof consists of R and c. A verifier computes $t_{hi} = h^{r_i} A^{c_i} \pmod{p}$ and $t_{gi} = (g^{1-c_i} V^{c_i B})^{(y^{r_i})}$ for $i = 1 \ldots \ell$, and checks whether (*) holds.

3.4 Analysis

There are two points to consider when discussing the security of the scheme. First, even if we assume that computing discrete logarithms and breaking ElGamal's public-key system is hard, we have to check whether computing v from both $V = g^v$ and the cipher-text (A, B), is also hard. Second, we have to make sure that the dealer cannot cheat in the verification protocol and that no "useful" information about v is given away. We can prove the following two propositions.

Proposition 1. *Under the assumption that computing discrete logarithms in G is infeasible, and that breaking the ElGamal cryptosystem is hard, computing v from g^v and $(h^\alpha, v^{-1}y^\alpha)$ is at least as hard as solving the Decision-Diffie-Hellman problem to the base h in $\mathbb{Z}_p^*$.*[2]

Sketch of Proof: Note that it is possible to decide whether the encrypted logarithm v is a quadratic residue in $\mathbb{Z}_p^*$, because the base h (and the public key y) is a quadratic residue, but that it remains difficult to break ElGamal's cryptosystem, i.e. to completely recover the encrypted message.

Assume that there is an efficient algorithm $\mathcal{P}$ that computes v on input $(g^v,\ h^z,\ h^\alpha,\ v^{-1}h^{\alpha z})$ with a non-negligible probability ε over all $v \in \mathbb{Z}_q^*$, and $z, \alpha \in \mathbb{Z}_q^*$. We show how to use $\mathcal{P}$ to decide whether a given triple (A, B, C) of elements in $\langle h \rangle$, is a Diffie-Hellman triple, i.e. whether $C = h^{\log_h A \cdot \log_h B} \pmod p$.

First, we need a method to randomize (A, B, C). Therefore, we choose $\rho \in \mathbb{Z}_q^*$, and $\sigma, \tau \in \mathbb{Z}_q$ at random and calculate $(\bar{A}, \bar{B}, \bar{C}) = (A^\rho h^\sigma, Bh^\tau, C^\rho A^{\rho\tau} B^\sigma h^{\sigma\tau})$. Since q, the order of h in $\mathbb{Z}_p^*$, is prime, it can easily be shown that the triple $(\bar{A}, \bar{B}, \bar{C})$ is a random Diffie-Hellman triple if (A, B, C) is a Diffie-Hellman triple, and a random non-Diffie-Hellman triple, otherwise.

Now, we randomly choose $v \in \mathbb{Z}_q^*$ and run $\mathcal{P}$ on input $(g^v, \bar{A}, \bar{B}, v^{-1}\bar{C})$. The probability that $\mathcal{P}$ returns v depends on whether (A, B, C) is a Diffie-Hellman triple or not:

- If (A, B, C) is a Diffie-Hellman triple then $\mathcal{P}$ returns v with probability ε.
- If (A, B, C) is not a Diffie-Hellman triple then the probability that $\mathcal{P}$ returns v is negligible. Let us assume on the contrary that $\mathcal{P}$ returns v with a non-negligible probability γ. Then the discrete logarithm of any $Y \in G$ can be computed by repeatedly running $\mathcal{P}$ on input $(Yg^\rho, h^\sigma, h^\tau, t)$ with $\rho \in_R \mathbb{Z}_p$, $\sigma, \tau \in_R \mathbb{Z}_q$, and $t \in_R \mathbb{Z}_p^*$ until $\mathcal{P}$ returns $\rho + \log_g Y \pmod p$. Because the probability that $t/(\rho + \log_g Y) \pmod p \in \langle h \rangle$ is approximately $1/2$, the expected number of repetitions is $2/\gamma$.

After sufficiently many repetitions, a decision on whether (A, B, C) is a Diffie-Hellman triple can be made with arbitrarily small probability of error.

Proposition 2. *The prover in the interactive protocol in Section 3.3 can successfully cheat with a probability of at most 2^{-K}. The protocol is perfectly zero-knowledge.*

Sketch of Proof: It can easily be seen that if in one round both challenges, $c = 0$ and $c = 1$, can correctly be answered then the claim holds, i.e. the logarithm of A to base h is equal to the double logarithm of V^B to the bases g and h. So if the claim does not hold, i.e.the two logarithms are different, a cheating prover can prepare t_g and t_h for only one challenge and will therefore be caught at cheating with probability $1/2$ in this round.

Zero-knowledgeness can be shown using standard techniques for constructing a simulator. □

[2] See [3] for a discussion of the Decision-Diffie-Hellman problem.

4 PVSS for Sharing e-th Roots

Methods similar to those presented in the previous Section can be used to share an *e-th* root of an element in a group $\mathbb{Z}_n^*$, where the factorization of n is unknown. For example, n and e could be the public parameters of a Fiat-Shamir [10] or a Guillou-Quisquater [14] signature scheme. A verifiable sharing scheme with general access-structure can be constructed in a similar way as described in Section 3.2; for the construction of a threshold scheme see [11]. What remains to show is an encryption scheme that allows to efficiently prove that a cipher-text contains the *e-th* root of a given element.

4.1 Verifiable Encryption of e-th Roots

Let $g \in \mathbb{Z}_n^*$ be a public value of large order. Each participant randomly chooses a secret key $z \in \mathbb{Z}_n$ and computes the corresponding public key $y = g^z \pmod{n}$.

A sender can now encrypt a value $m \in \mathbb{Z}_n^*$ by randomly choosing $\alpha \in \mathbb{Z}_n$ and calculating

$$A = g^{\alpha} \pmod{n}, \text{ and } B = m \cdot y^{\alpha} \pmod{n}.$$

The recipient can easily obtain the *e-th* root of M by calculating

$$m = B/A^z \pmod{n}.$$

With the following interactive protocol the sender can prove that the pair (A, B) encrypts the *e-th* root of $M = m^e \pmod{n}$ $(\epsilon > 0)$.

Prover		**Verifier**
repeat K times:		
$w \in_R \{0, \ldots, \lceil 2^{\ell} n^{1+\epsilon} \rceil\}$ $t_g = g^w \pmod{n}$ $t_y = y^{ew} \pmod{n}$	$\xrightarrow{t_g,\, t_y}$	
		$c \in_R \{0, \ldots, 2^{\ell} - 1\}$
	$\xleftarrow{c}$	
$r = w - c \cdot \alpha$ (calculation in $\mathbb{Z}$)		
	$\xrightarrow{r}$	
		$t_g \stackrel{?}{=} g^r A^c \pmod{n}$ $t_y \stackrel{?}{=} y^{er} (B^e/M)^c \pmod{n}$

For a non-interactive proof we need a cryptographically strong hash-function $\mathcal{H}_{\ell} : \{0,1\}^* \to \{0,1\}^{\ell}$. The sender chooses a random $w \in \{0, \ldots, \lceil 2^{\ell} n^{1+\epsilon} \rceil\}$ and computes $t_g = g^w \pmod{n}$, $t_y = y^{ew} \pmod{n}$, $c = \mathcal{H}(M||A||B||t_g||t_y)$, and $r = w - c \cdot \alpha$. The resulting proof is (r, c); verification is straightforward. If M

and the cipher-text (A, B) are included, the whole share has a length of only $2\ell + (4 + \epsilon) \cdot \log_2 n$ bits. For a "practical" scheme we recommend to choose $n > 2^{750}$, $\ell > 80$ and $\epsilon \approx \frac{1}{5}$.

4.2 Analysis

As in Section 3.4, we have to consider the security of the encryption scheme and the security of the verification protocol. Unfortunately, a statement similar to proposition 1 is difficult to prove. This is mainly because the order of g is not prime and therefore a good randomization of non-Diffie-Hellman triples is not possible anymore. If we required that the order of g is prime, then the security of the scheme could only be proved (in the manner of proposition 1) for messages m that belong to the subgroup generated by g.

For the security of the verification protocol, we can prove the following proposition:

Proposition 3. *The prover in the interactive protocol in Section 4.1 can successfully cheat with a probability of at most $2^{-K\ell}$. The protocol is statistically zero-knowledge if $\ell = \mathcal{O}(\log \log n)$.*

Sketch of Proof: The first claim can be proved in a similar manner as for proposition 2. For proving zero-knowledgeness we construct a simulator that first randomly chooses $r \in \{0, \ldots, \lceil 2^\ell n^{1+\epsilon} \rceil\}$, guesses $c \in \{0, \ldots, 2^\ell - 1\}$, computes t_g and t_y, and then checks whether c was correctly guessed or not (according to the verifier's strategy). For $\ell = \mathcal{O}(\log \log n)$ this simulator runs in expected polynomial time. It remains to show that the output of the simulator and the output of the protocol are statistically indistinguishable (see [13] for definition). $\square$

Acknowledgments

Many thanks to U. Maurer, D. Bleichenbacher, C. Cachin, J. Camenisch, and the anonymous referees for their useful comments.

References

1. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.
2. B. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference 1979*, volume 48 of *American Federation of Information Processing Societies Proceedings*, pages 313–317, 1979.
3. S Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, Amsterdam, 1993.
4. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, Oct. 1988.

5. E. Brickell, P. Gemmell, and D. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 457–466. ACM, 1995.
6. J. Camenisch, J.-M. Piveteau, and M. Stadler. An Efficient Fair Payment System. To appear in *Proc. 3rd ACM Conference on Computer and Communications Security*, 1996.
7. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 383–395, 1985.
8. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.
9. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, July 1985.
10. U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1:77–94, 1988.
11. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 427–437, 1987.
12. A. Fiat and A. Shamir. How to prove yourself: Practical solution to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.
13. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th ACM Symposium on Theory of Computing (STOC)*, pages 291–304, 1985.
14. L. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In *Advances in Cryptology – EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer-Verlag, 1988.
15. M. Jakobsson and M. Yung. Revokable and Versatile Electronic Money. To appear in *Proc. 3rd ACM Conference on Computer and Communications Security*, 1996.
16. S. Micali. Fair cryptosystems. Technical Report TR-579.b, MIT, November 1993.
17. NIST. Clipper chip technology, 30 April 1993.
18. T. Pedersen. Distributed provers with applications to undeniable signatures. In *Advances in Cryptology – EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 221–242. Springer-Verlag, 1992.
19. C. Schnorr. Efficient identification and signature for smart cards. In *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer-Verlag, 1990.
20. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
21. M. Stadler, J.-M. Piveteau, and J. Camenisch. Fair blind signatures. In *Advances in Cryptology – EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219. Springer-Verlag, 1995.