# A PVSS as Hard as Discrete Log and Shareholder Separability

Adam Young[1] and Moti Yung[2]

[1] Columbia University, New York, NY, USA.
ayoung@cs.columbia.edu
[2] CertCo, New York, NY, USA.
moti@cs.columbia.edu

**Abstract.** A Publicly Verifiable Secret Sharing (PVSS) scheme allows a prover to verifiably prove that a value with specific properties is shared among a number of parties. This verification can be performed by anyone. Stadler introduced a PVSS for proving that the discrete log of an element is shared [S96], and based the PVSS on double-decker exponentiation. Schoenmakers recently presented a PVSS scheme that is as hard to break as deciding Diffie-Hellman (DDH) [Sch99]. He further showed how a PVSS can be used to improve on a number of applications: fair electronic cash (with anonymity revocation), universally verifiable electronic voting, and software key escrow schemes. When the solution in [Sch99] is used for sharing a key corresponding to a given public key, the double-decker exponentiation method and specific assumptions are still required. Here we improve on [Sch99] and present a PVSS for sharing discrete logs that is as hard to break as the Discrete-Log problem itself, thus weakening the assumption of [Sch99]. Our solution differs in that it can be used directly to implement the sharing of private keys (avoiding the double decker methods). The scheme can therefore be implemented with any semantically secure encryption method (paying only by a moderate increase in proof length). A major property of our PVSS is that it provides an algebraic decoupling of the recovering participants (who can be simply represented by any set of public keys) from the sharing operation. Thus, our scheme diverts from the traditional polynomial-secret-sharing-based VSS. We call this concept *Separable Shareholders*.

## 1 Introduction

Secret sharing schemes were introduced to enable the distribution of trust among several participants. In a secret sharing scheme a secret is split into several pieces and is shared among several participants. Only when the shares are put together, or in the case of threshold sharing schemes when some subset of the shares are put together, is the secret reconstructed. To protect against cheating participants, the notion of Verifiable Secret Sharing was introduced [CGMA,F85]. In VSS, a verification protocol allows the participants to verify that the unique secret can be reconstructed when needed. A property of a VSS which was emphasized by Stadler [S96] is that "not only the participants, but anyone can verify that

K. Kim (Ed.): PKC 2001, LNCS 1992, pp. 287–299, 2001.

the secret is shared properly". When a VSS scheme has this public verifiability property, it is called a Publicly Verifiable Secret Sharing (PVSS) scheme.

The original PVSS needed the double-decker discrete log assumption [S96]. Other special assumptions for PVSS schemes were given in [FO98] (in particular a special RSA assumption which allows partial recovery), whereas [Sch99] managed to reduce the assumption of a discrete log PVSS to regular Decision DH (DDH). Here we manage to reduce the required assumptions even further, which simplifies the scheme and enhances its usability and availability as we will explain. It is worth noting in this context the interesting work of [PS00], where they develop a nice approach to short non-interactive proofs. However this is a new approach of a somewhat different flavor and obviously of different optimization goals (i.e., their very short elegant proof scheme does not assure asymptotic inverse exponential reduction in security, but relies on the state of the art of certain algorithms. Also, the recovery of secrets in their case may be delayed).

We note that a random oracle assumption for making proofs transferable (NIZK a la Fiat-Shamir) and secure encryption assumptions for implementing secure authenticated channels are also employed whenever a PVSS scheme is designed. For our scheme it is possible to use any probabilistic encryption. Our new scheme enables the use of schemes other than these based on DDH. In particular, for DH based schemes, a practical scheme which is based on a computational Diffie-Hellman assumption, namely, the partial-trapdoor property of DH (and a random oracle) but that *does not require* the DDH assumption can be employed [FO99,P00].

Several useful applications can be built using PVSS schemes. Among the applications of PVSS are voting [Sch99], anonymity-revocation in e-cash systems (e.g., [FTY96]), escrow systems [Mi,S96,VT,YY98], and certified e-mail [Mi98].

A major general advantage of our design (which we bring to the above applications) is a separation of the structure and organization of the recipients (share-holders, election officials, authorities, etc.) from the rest of the world. Thus, the share-holder group can be viewed as an organization which can be managed internally and be presented implicitly (keeping the internal structure hidden) or explicitly (specified access structure and keys) to the rest of the world. This allows a dynamic (and also parallel) share-holder organization. The separation of internal organizational workings and the external world (and its importance to managing evolving commercial entities and consortium bodies) was put forth in [FY99] for general PKI, and, in the context of election schemes it was implemented in [CGS97]. In particular, [FY99] suggests to employ "proactive key maintenance" methods to change the share holder group, while keeping the shared information intact. Our constructions when used with implicitly shared keys, support such operations.

## 2 The Definition of a PVSS

We will now present the informal definition of a PVSS taken directly from section 2 of [S96] (here $s$ is the value being shared).

Let $\mathcal{A}$ be a monotone access structure. Since $\mathcal{A}$ is a monotone access structure, it follows that if $A \in \mathcal{A}$ and $A \subseteq B$, then $B \in \mathcal{A}$.

**Definition 1.** *A PVSS consists of a dealer, n participants $P_1$,...,$P_n$ such that each has a public encryption function $E_i$ and such that each has a corresponding secret decryption function $D_i$, a monotone access structure $\mathcal{A} \subseteq 2^{\{1,...,n\}}$, and algorithms Share, Recover, and PubVerify which operate as follows:*

*Share: The dealer uses the public encryption functions to distribute the shares by calculating $S_i = E_i(s_i)$ for $1 \leq i \leq n$. The dealer then publishes each share $S_i$.*

*Recover: If a group of participants want to recover the secret, they run Recover, which has the property that $\forall A \in \mathcal{A}$: $Recover(\{S_i | i \in A\}) = s$, and that for all $A \notin \mathcal{A}$ it is computationally infeasible to calculate s from $\{S_i | i \in A\}$.*

*PubVerify: To verify the validity of all encrypted shares, PubVerify is run by any inquiring party. This algorithm has the property that $\exists u \ \forall A \ \in 2^{\{1,...,n\}}$:*

$$(PubVerify(\{S_i | i \in A\}) = 1) \Rightarrow Recover(\{D_i(S_i) | i \in A\}) = u$$

*and $u = s$ if the dealer was honest.*

A PVSS is called non-interactive if PubVerify requires no interaction with the dealer at all.

**Properties of PVSS:**

**Completeness:** We say that a PVSS is **complete** if whenever the dealer is honest (and the (unique) value for $s$ is recoverable by the participant(s)), the verifier accepts the prover's proof as valid (with overwhelming probability).

**Soundness:** A PVSS is **sound** if whenever the unique $s$ is not recoverable, the verifier accepts the prover's proof only with negligible probability.

The last two properties are important for the notion of a PVSS to be correct. The necessary notion of completeness was omitted from the informal definition of [S96]. We note that the above properties put together also imply that the dealer which encrypts with a group's key (even though it encrypts to a group of servers which it does not even interact with directly), does not have to use chosen ciphertext secure encryption (as our complete proof below demonstrates). This is in contrast to other recent applications of proving properties of encryption.

**Secrecy:** Finally, another property of a PVSS is **secrecy**, which means that any group not in the access structure should not be able to retrieve $s$ (but perhaps with negligible probability) given the public output of Share.

Observe that the above definition does not state that $s$ must be of any particular form. For example, this definition does not state that $s$ must be a private key, and that PubVerify must be able to verify that $s$ is the private key corresponding to some public key. In fact we may distinguish between the cases in

which a value, private key, or encryptions under a shared public key are recovered. These are interesting sub-cases which are required for various applications of PVSSs [1].

Also, the monotone access structure in the original definition may be extended to include threshold schemes [Sch99] (see [D92,FY98]). In fact, we will employ any semantically secure encryption function by the participants, which will enhance the applicability of our scheme.

## 3 PVSS for Discrete Logs Based on the DL Problem

We will now present a PVSS for discrete logs (DL) which is based on the difficulty of computing discrete logs. This contrasts with the PVSS scheme in [Sch99] which assumes the difficulty of Deciding Diffie-Hellman.

The following are the cryptographic primitives that are used in our system. *enc* is a a semantically secure probabilistic public key encryption algorithm that takes three arguments, $m$, $s$, and $PUB$. Here $m$ is a message to be encrypted, $s$ is a randomly chosen string to make the encryption probabilistic, and $PUB$ is the public key of the participant (or shared among a set of participants or created by individual public keys of participants). Thus, $c = enc(m, s, PUB)$ is the ciphertext of the message $m$. Let *dec* be the corresponding decryption function. Thus, $m = dec(c, PRIV)$, where $PRIV$ is the private key corresponding to $PUB$. It could be that $PRIV$ is shared distributively, in which case $m = dec(c, PRIV_1, PRIV_2, ..., PRIV_m)$. This can model either a threshold scheme or a polynomial-size monotone access structure composed of nested encryptions of the value by the keys of members of groups in the access structure.

The prover who is the dealer generates a private value $x$ and its corresponding public commitment $y = g^x \ mod \ p$. We can insist that $p$ is a safe prime and that $g$ has order $p-1$, or we can insist that $g$ has order $q$ where $q$ is a large prime dividing $p-1$. For this section we will w.l.o.g. assume that $g$ has order $p-1$. Informally, the system then works as follows. The dealer commits to knowing two shares that sum to the dealer's private exponent $x \ mod \ p-1$. This commitment is performed using the envelope method in which additive pieces of the secret are committed to separately [2]. The method employs a public homomorphic commitment for each piece, and a further commitment to a piece which is performed using the arbitrary probabilistic encryption function *enc* under the participants public keys(s) (shared or explicitly combined key(s) that is). Thus, *enc* provides a semantically secure encryption relative to the public homomorphism, and hence

[1] Of course, sharing a public-key implies sharing a value (which can be encrypted under that key) and sharing the key implies recovery of messages encrypted under that key as well, and we leave as open other implications and separations which may be needed in applications.

[2] The envelope method is different from the usual polynomial sharing which is extensively used; since we are going to use additional shared encryption (owned by the recovery participants) we found that there is no need to employ, prior to the encryption, explicit sharing techniques like polynomial sharing schemes.

provides a secure commitment with respect to the verifier, who can be anyone including the participants. A total of $M$ additive envelope pairs are committed to using *enc*. Using challenge bits, the dealer is forced to reveal a share, not of his choosing, for each pair of additive shares. So, the system in some sense constitutes a proof that at least one pair of shares that sum to $x$ has been committed to under *enc*. Thus, with overwhelming probability, the transcript of the proof itself can be used by the participant(s) to recover $x$.

We will next review the algorithm in detail.

**Share:** The following is the non-interactive (transferable) transcript generation algorithm based on H being a random oracle. It is this transcript that forms the verifiable encryption of the prover's secret $x$.

1. $P = ()$
2. for $i = 1$ to $2k(n)$ do
3. $\quad r_i \in_R Z_{p-1}$
4. $\quad$ choose two random strings $s_{i,1}$ and $s_{i,2}$ for use in *enc*
5. $\quad Q_i = g^{r_i} \ mod \ p$
6. $\quad C_{i,1} = enc(r_i, s_{i,1}, PUB)$
7. $\quad C_{i,2} = enc(r_i - x \ mod \ p - 1, s_{i,2}, PUB)$
8. $\quad$ add $(Q_i, C_{i,1}, C_{i,2})$ to the end of $P$
9. val = H($P$)
10. set $b_1, b_2, ..., b_{2k(n)}$ to be the $k(n)$ least significant bits of val, where $b_i \in Z_2$
11. for $i = 1$ to $2k(n)$ do
12. $\quad w_i = r_i - b_i x$
13. $\quad z_i = (w_i, s_{i,j})$ where $j = 1 + b_i$
14. $\quad$ add $z_i$ to the end of $P$

Thus, $P = ((Q_1, C_{1,1}, C_{1,2}), ..., (Q_{2k(n)}, C_{2k(n),1}, C_{2k(n),2}), z_1, ..., z_{2k(n)})$. Here $P$ denotes the non-interactive proof, or transcript. Note that the $b_i$'s can be recovered from $P$. The algorithm outputs $(y, x, P)$ to the dealer.

**PubVerify:** To verify that $x$ is recoverable by the participant(s) who own(s) $y$, the verifier takes $y$, the corresponding $P$, and the public key $PUB$. The verifier first checks that $y < p$. The verifier checks that all of the values in $P$ lie in the correct sets. The verifier also checks that the values $C_{i,j}$ for all $i$ and $j$, do not contain any repetitions. The verifier checks that none of the $Q_i$ for all $i$ are repetitious. If any of these verifications fail, then false is returned (the prover can easily avoid repetitions by checking that the chosen values are not repeating which is a negligible probability event for the honest prover). The verifier then computes $b_1, b_2, ..., b_{2k(n)}$ in the same way as in the "share" computation. For $i = 1$ to $2k(n)$, the verifier verifies the following things:

1. $enc(w_i, s_{i,j}, PUB) = C_{i,j}$ where $j = 1 + b_i$
2. $Q_i/(y^{b_i}) \ mod \ p = g^{w_i} \ mod \ p$

The verifier concludes that $x$ is recoverable as long as all the verifications pass and as long as both 1 and 2 above are satisfied for $1 \leq i \leq 2k(n)$.

**Recover:** When the shareholder(s) obtain(s) the non-interactive proof $P$ all values are decrypted and the second cleartext value is subtracted from the first cleartext value mod $p-1$ for each pair to obtain a value. To verify that such a value is $x$, the shareholder(s) raise(s) $g$ to this value mod $p$ and compares it with $y$. We remark that the ability to do this in a shared fashion depends on the algorithms used for *enc* and *dec*. Some schemes may use threshold encryption, whereas general semantically secure algorithms can always support a shared *dec* algorithm in which decryption is performed by each participant in turn, giving the result to the next participant for decryption. The trust model in the later case has to be such that the last participant to decrypt has to be trusted to share its information. Verification of the value is public due to the public homomorphic commitments.

## 4 Security and Correctness

The non-interactive PVSS for sharing $x$ was constructed based on a 3-round atomic computational ZKIP with error probability 1/2. To see this, note that it directly corresponds to the following interactive 3-round protocol[3]:

1. For $i = 1$ to $k(n)$ do:
2. P chooses $r_i \in_R Z_{p-1}$
3. P chooses $s_{i,1}$ and $s_{i,2}$ randomly for use in enc
4. P computes $Q_i = g^{r_i} \ mod \ p$
5. P computes $C_{i,1} = enc(r_i, s_{i,1}, PUB)$
6. P computes $C_{i,2} = enc(r_i - x \ mod \ p - 1, s_{i,2}, PUB)$
7. P sends $(Q_i, C_{i,1}, C_{i,2})$ to V
8. V sends $b_i \in_R \{0, 1\}$ to P
9. P computes $w_i = r_i - b_i x \ mod \ p - 1$
10. P sends $z_i = (w_i, s_{i,j})$ where $j = 1 + b_i$ to V
11. V verifies that:
12. $enc(w_i, s_{i,j}, PUB) = C_{i,j}$ where $j = 1 + b_i$
13. $Q_i/(y^{b_i}) \ mod \ p = g^{w_i} \ mod \ p$

Here $n$ be a security parameter and $k(n) = \omega(log \ n)$ is given. The proof achieves error $2^{-k(n)}$. Soundness of the above protocol can be seen by the fact that if $P$ does not know $x$ then in a given round $P$ can respond to only one possible outcome, otherwise $P$ can compute discrete logarithms. It differs from standard zero-knowledge proofs in that, in addition to a commitment value being sent in the first round (i.e., using $Q_i$ to commit to the base $g$ logarithm of $Q_i$ mod $p$), the prover sends two semantically secure encryptions (which must be consistent with the commitment in $Q_i$ and $x$). The first of these encryptions

[3] We comment that an interactive variant which is based on parallel execution (using claw-free function commitments by the verifier) exists and that allows the entire proof to be conducted in a small (constant) number of rounds [GK,BMO,IS93,BJY97].

can be thought of as yet another commitment of the base $g$ logarithm of $Q_i$. The second encryption is a commitment to the base $g$ logarithm of $Q_i/y$. Since these are both semantically secure encryptions, they give nothing to a poly-time bounded adversary that the adversary couldn't compute himself. They are, in some sense, redundant commitments of $Q_i$ and $Q_i/y$ (they are useful since they provide recoverability by the participants). In the second round, the verifier sends a randomly chosen bit to the prover, as in standard zero-knowledge proofs. In the third round, the verifier either opens the commitment $r_i$ (for $Q_i$) or opens $r_i - x \ mod \ p - 1$ (for $Q_i/y$), as in standard zero-knowledge proofs. The only difference is that exactly one of the two semantically secure encryptions is also opened, and verified for consistency (to insure that "recover" works). In arguing zero-knowledge of the interactive procedure, $r_i$ is chosen at random and if $b_i$ is expected to be 0, $Q_i$ is computed as in the protocol. In this case $C_{i,1}$ is encrypted correctly and $C_{i,2}$ is formed by encrypting the string with each bit being zero (indistinguishable from a correct encryption). On the other hand, if $b_i$ is expected to be 1, then again $r_i$ is chosen at random, $C_{i,1}$ is made to be an encryption of zeros, and $C_{i,2}$ is made to encrypt $r_i$. In addition $Q_i = g^{r_i}y$ is published (so that the verification checks pass). Standard zero-knowledge, assuming commitment (encryption in our case) implies that the simulation can be conducted in expected polynomial time. Completeness follows a standard argument. Thus, given $y$ the above is a complete and sound and computational zero-knowledge (simulatable) interaction.

**Lemma 2.** *The PVSS scheme is computational zero-knowledge and therefore secure in the random oracle model (with error* $1/2^{k(n)}$*).*

*Proof.* (sketch). In section 5.2 of [BRa], a reduction is given that shows how to transform any three move ZKIP for $L \in NP$ with error probability 1/2 into a non-interactive ZK proof in the random oracle model. The reduction accounts for a set of envelopes being sent in the first round, and having some subset of the envelopes opened according to a randomly chosen challenge bit $b$. The reduction therefore applies to our protocol above for discrete logs. The transformation that is given in [BRa] is the same transformation that was used to obtain the non-interactive version given above. Since semantically secure encryptions yield nothing that can't be computed efficiently without the encryptions, the lemma then follows from the transformation. Note that since the transcript is zero-knowledge (simulatable) given $y = g^x$ then whatever is derivable from the public key $y$ about the private key $x$ without the transcript, can be derived with the transcript with related probability (the relation is given by the simulation argument given in the transformation). In particular, if $x$ can be derived from $y$ and the transcript by a party that does not hold $PRIV$ (namely, violation of secrecy), then if *enc* is semantically secure, we can use the successful derivation to break the discrete log assumption. QED.

We will now prove that the non-interactive PVSS for DL based keys is a proof of knowledge in the random oracle model. This will imply that the PVSS itself is, with respect to shares, complete and sound. This part does not need

the assumption that computing discrete logs is intractable but we still need the fact that the encryption is indeed a commitment scheme and we still need to use the random oracle model (for the protocol to be non-interactive and produce a transferable transcript). We do not give in this version a formal definition for a non-interactive proof to be a proof of knowledge in the random oracle model. However, in a nutshell, it is a continuation of the formalization of the Fiat Shamir [FS86] notion (which was formalized but not for proofs of knowledge in [BRa,PS96]). We would like that possible unpredictable different (forking of) values of a polynomial portion of the oracle answers imply the extractability of the witness value. We then extend extraction, to extraction by a third party who can recover the commitment. (Familiarity with proof of knowledge, extractors, and the random oracle model is assumed).

**Lemma 3.** *The non-interactive PVSS for DL based keys constitutes a proof of knowledge with knowledge error* $1/2^{k(n)}$.

*Proof.* It is easy to see (similar to the zero-knowledge proof being complete) that the non-triviality condition holds. We will now consider the validity condition. The common input $\alpha$ is $y = g^x \ mod \ p$. The auxiliary input (witness) $\beta$ is $x \ mod \ p-1$. We have that $x_i = (Q_i, C_{i,1}, C_{i,2})$, $y_i = z_i$, $t = 2k(n)$, and the $b_i$'s in the proof are the same as the $b_i$'s in the definition. Suppose that P makes a total of $T(n)$ oracle queries when given $H$ and $H'$ (e.g., in an attempt to fool V). If the prover doesn't know a witness then the prover will convince V of the validity of the proof with probability at most $T(n)2^{-2k(n)}$ which is at most $1/2^{k(n)}$ for sufficiently long n (see [BRa] section 5.2). Thus, the knowledge error is at most $1/2^{k(n)}$. It follows that $p(\alpha)$ is at least $1 - 1/2^{k(n)}$.

Let $T = P_{\alpha,\beta,r}(H)$ and $T' = P_{\alpha,\beta,r}(H')$ Consider the following knowledge extractor $K$. Suppose that in round $i$, $b_i$ in $T$ is 0 and $b_i$ in $T'$ is 1. $K$ then subtracts the $w_i$ in $T'$ from the $w_i$ in $T$ mod $p-1$ to get $x_c$, a candidate value for $x$. The operation of $K$ when the bits are inverted is similar.

We will now give a lower bound on $K$'s probability of extracting a witness from $T$ and $T'$ (we won't derive an exact probability, since using $T(n)$ oracle queries, P may always try to make the first half of the $b_i$'s 0 to try to fool the prover, for example). Assuming no extra oracle queries are made, with probability $1/2$ we have that $b_i$ in $T$ equals $b_i$ in $T'$, since the $b_i$'s are chosen randomly for both $T$ and $T'$ ($H$ and $H'$ serve as honest verifiers). So, with probability $1/2$, the $b_i$'s in each transcript differ and the knowledge extractor can extract $x$ from both de-committed values. To see this recall that P for both transcripts is using: the same common-input $\alpha$, the same auxiliary input $\beta$, and *the same random tape* $r$. The probability that $x$ isn't extracted in any of the $2k(n)$ rounds is $2^{-2k(n)}$. Now consider the case where P makes $T(n)$ additional oracle queries. It can be shown that the probability that $x$ isn't extracted is at most $T(n)2^{-2k(n)}$. For sufficiently large $n$ this probability is at most $2^{-k(n)}$. Thus, the probability that the witness $s = x$ is extracted by $K$ is at least $1 - 2^{-k(n)}$. Taking this worst case value, along with the worst case values for $p(\alpha)$ and the knowledge error, we get the claimed knowledge error.a QED.

**Theorem 4.** *The PVSS above is a complete, sound, and secure protocol.*

Completeness holds since any decrypted pair can give the correct secret that is shared. The proof of knowledge (extraction) which is done by verifying the commitments (which are in fact encryptions openable by the owners of the encryption's private keys) implies that with the same probability of extraction (as in Lemma 2), the owner of the encryption method can decrypt both envelopes of the correct shared secret (and verifies its value against its public witness), and the owner is fooled only with a negligible probability (that of the knowledge error). This implies soundness. Lemma 1, in turn, implies secrecy.

## 5 Applications

**Shareholder Separability:**
The simplified PVSS here treated the encryption keys of the shareholding authorities as a given black box. This enables many applications in an extended setting where the public key schemes of authorities/shareholders/agencies are managed and organized separately from the users who only have access to an agency's public key; various authorities, in fact, may have different types of public keys. This principle of separation of management of various roles is key in evolving organizations. Cryptographic designs in critical commercial and financial settings should follow such principles, as advocated in [FY99] in the context of PKI. The structure and composition of the shareholder group can be changed using "proactive maintenance" [OY91] and the secret is not revealed while the shareholders change as discussed in [FGMY].
**E-cash with anonymity revocation:**
Various schemes have been used to implement e-cash where the authorities are off-line and the user performs a proof of encryption with the trusted authority keys (e.g. [FTY96]). In these schemes the trustees' key is a shared ElGamal key. Since our PVSS decouples the receivers' key (it can be any public key, have any access structure, and have any organization), we can extend the above schemes by allowing the user to commit to the value of the coin based on the discrete log problem while encrypting using any scheme. The PVSS becomes a generalized "indirect discourse proof." This may be a significant extension which enables the development of the structure of the trustees and their organizational and operational changes independently from the underlying scheme. The changes may be done implicitly or explicitly (notions described in detail in [FY99]). If an explicit change is performed, the trustees' keys are publicly changed and the users have to be notified to spend or replace their coins for the coins to maintain their value. An implicit change does not change the external view of the trustees and seems preferable in this setting.
**Universally-verifiable secure ballot election:**
We can use our PVSS to obtain a simplified scheme for secure ballot election (in the setting of [CF85]) or with distributed tallying authorities (as in [BY86]). Here we assume that the exponents are taken from a prime order subgroup generated by $g$ (say $p = 2q + 1$, where $q$ is a large prime and $g$ generates the quadratic

residues $G_q$). Voter $i$ commits to a random $G_i = g^s$ and to $H_i = y^s$ or $y^{(s+1)}$, depending on whether it casts 0 or 1, respectively (the discrete log of $y$ mod $g$ is globally unknown, the security here relies on the semantic security of ElGamal in the subgroup which is equivalent to DDH [TY,NR]). The voter proves (in NIZK) the relationship between $G_i$ and $H_i$ (see [Sch99]). It then PVSS's the value of $s$ in $G_i = g^s$. The tallying authorities can recover $s$ and the ballot ($s$ or $s + 1$) for each voter but will keep it secure. The voting is robust (each user's PVSS proof is publicly verifiable) and can be made independent of other users (by changing the hashing procedure for users based on the unique user ID). The authorities can present and prove the correctness of the results. Everyone can compute the product of the values of $G = \prod_i G_i$ and that of $H = \prod_i H_i$. The discrete logs of $G, H$ and their difference $mod\ q$ (the exact tally) is available to the authorities, who can claim the result and present a NIZK proof of knowledge of this value based on the public availability of $G, H$ and the result (and the private knowledge of the discrete log values). Of course the exponent additive group size, $q$, is larger than the number of voters which, together with the NIZK of the users being verified, prevents wrap-arounds of the difference above.

Once again, the keys and organization of the tallying authorities and their structure is independent of the ballot construction. This has some advantages such as: a number of parallel tallying authorities can be easily implemented (by encrypting in parallel with their keys), authorities organized as a general linear access structure is possible, dynamic changes in the tallying authorities between and during elections [FY99] is possible, etc. A somewhat more specific separation was given already in the scheme of [CGS97].

**Software key escrow applications:**

In [YY98] a model and solution for key recovery in the context of a public-key infrastructure was given and implementation based on double-decker exponentiation was given. In this model, each user is responsible for escrowing his or her own private key and is responsible for constructing a proof to this effect. Such a user is granted a digital certificate by a CA only if a public key, an encryption of the corresponding private key, and a proof that the encryption is correct is supplied to the CA during key certification (and only during certification). The PVSS scheme can be used to solve this problem as follows. The user computes the public key $y = g^x\ mod\ p$ where $p$ is cryptographically secure (e.g. $p - 1$ is a multiple of a large prime). The prover, who is the dealer in the PVSS scheme then shares the private key $x$ among the participants which are the escrow authorities (just by knowing the keys of the authorities which are public). For verification, since the CAs know the shared public key of the participants, the user can send $y$ along with the proof to any CA for key certification. The CA is thus the verifier in the PVSS. The recover procedure is run by the authorities given the transcript from the CA. The authorities recover the private key (or even better they recover encryptions of session keys encrypted under the user's public key without ever recovering the user's private key itself (as advocated in [LWY95,FY95]) which is possible in various threshold cryptography settings).

In contrast, in [Sch99] a similar key escrow solution based on [YY98] was given that makes use of a value $f$ which generates a high order subgroup of $Z_p$, and which uses a fixed element $g$ with high order in $Z_p^*$. In that solution the value $C_0 = g^s$ is published (in the distribution protocol of section 3 in that paper). Thus, the escrow solution is secure only if computing discrete logs in a subgroup is intractable (as well as the DH assumption). Note that the parameters $g$ and $h$ require that $p = 2tq + 1$ and $q = 2wr + 1$ where $p$, $q$, and $r$ are primes and $t$ and $w$ are positive integers. Also, the public key in that solution is $H = f^{(G^s)}$, where $f$ is an appropriate generator (as in [S96]). Since both $H$ and $C_0$ are public, this escrow solution makes a cryptographic assumption above and beyond the new PVSS, and this assumption is missing from the paper. The cryptographic assumption is that given $(H, C_0)$ it is intractable to recover $G^s$ (and therefore $s$ too). Note that the exact same assumption was made in [YY98], where it was refereed to as "problem 1" (which is a DH-Dlog combined problem). This is *not* the standard DH assumption. It is important to distinguish between the assumptions made for the security of the PVSS *protocol*, and the assumptions made for the security of the published *trapdoor values*. This is especially true when a PVSS is used for software key escrow, since the requirements for a secure software key escrow solution exceed the requirements of a PVSS (e.g., sharing a public key, or recovering only values encrypted under that key; we can achieve both in the last application).

**Certified mail:**

When an escrow system and signature scheme are in place a simple "optimistic" certified mail system is possible with an off-line post-office (an idea due to Micali [Mi98]). The sender commits to the encrypted mail and the mail key, signs this message, and sends it to the receiver together with a proof of that everything was constructed correctly, which in our case is just the PVSS escrowing the message key (as above) under the post-office key. The receiver sends back a receipt acknowledging the above message and signs this message. The sender then sends the mail decrypted (by sending the key). If the last message is not received promptly (under an agreeable definition of "promptly") the receiver gets the post-office involved. The PVSS assures the receiver that the encrypted key is recoverable by the post office (under any organization of the post office agents).

## References

BJY97. M. Bellare, M. Jakobsson, M. Yung. Round-Optimal Zero-Knowledge Arguments Based on Any One-Way Function. Eurocrypt'97 pp. 280–305.

BRa. M. Bellare, P. Rogaway. Random Oracles are Practical In *ACM CCCS '94*.

BMO. M. Bellare, S. Micali, R. Ostrovsky. Perfect Zero-Knowledge in Constant Rounds. In *ACM STOC '90*.

CF85. J. Cohen (Benaloh) and M. Fischer, *A robust and verifiable cryptographically secure election scheme*, FOCS 1985, pp. 372–382.

BY86. J. C. Benaloh and M. Yung, *Distributing the Power of a Government to Enhance the Privacy of Voters*, PODC 1986, pp. 52-62.

CGS97. R. Cramer, R. Gennaro and B. Schoonmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In Eurocrpt'97, pages 103–118.

CGMA. B. Chor, S. Goldwasser, S. Micali, B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *FOCS '85*.

D92. Y. Desmedt. Threshold cryptosystems. AUSCRYPT '92, 3–14.

F85. P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *FOCS '87*.

FGMY. Y. Frankel, P. Gemmell, P. MacKenzie, M. Yung. Optimal Resilience Proactive Public Key Systems. In *FOCS '97*.

FS86. A. Fiat, A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. Crypto'86 pages 186–194.

FTY96. Y. Frankel, Y. Tsiounis, M. Yung. Indirect Discourse Proofs: Achieving Efficient Fair Off-Line Cash. In *Advances in Cryptology—Asiacrypt '96*.

FY95. Y. Frankel, M. Yung. Escrow Encryption Systems Visited: Attacks, Analysis and Designs. In *Advances in Cryptology—Crypto '95*, pages 222–235.

FY98. Y. Frankel and M. Yung. Distributed public-key cryptosystems. In *Advances in Public Key Cryptography—PKC '98*, volume 1431 LNCS, 1–13.

FY99. Y. Frankel, M. Yung. Cryptosystems Robust against "Dynamic Faults" Meet Enterprise Needs for Organizational "Change Control." In *Financial Cryptology 99*.

FO98. E. Fujisaki and T. Okamoto, A Practical and Provably Secure Scheme for Publicly Verifiable Secret Sharing and Its Applications. Eurocrypt'98.

FO99. E. Fujisaki and T. Okamoto, Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Crypto'99.

GK. O. Goldreich, A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3), pp. 167–190, 1996.

GM. S. Goldwasser, S. Micali. Probabilistic Encryption. In *JCSC '84*.

IS93. T. Itoh, K. Sakurai. On the complexity of constant round ZKIP of possession of knowledge. In *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, vol. E76-A, No. 1, Jan. 1993.

Luby. M. Luby. Pseudorandomness and its Cryptographic Applications. Princeton Press.

LWY95. A. Lenstra, P. Winkler, Y. Yacobi. A Key Escrow System with Warrant Bounds. In *Advances in Cryptology—Crypto '95*, pages 197–207.

Mi. S. Micali. Fair Public-Key Cryptosystems. Crypto'92, pp. 113–138.

Mi98. S. Micali. Certified E-mail with Invisible Post Offices. Weizmann Institute Workshop, talk, June 98.

NR. M. Naor, O. Reingold, Efficient Cryptographic Primitives based on Decision Diffie-Hellman. In *FOCS '97*.

OY91. R. Ostrovsky and M. Yung, *How to withstand mobile virus attacks*, PODC 1991, pp. 51-61.

P00. D. Pointcheval, Chosen-Ciphertext Security for Any One-Way Cryptosystem. PKC'00.

PS96. D. Pointcheval, J. Stern. Security Proofs for Signature Schemes. Eurocrypt'96.

PS00. G. Poupard, J. Stern. Fair Encryption of RSA keys, Eurocrypt'00.

S96. M. Stadler. Publicly Verifiable Secret Sharing. Eurocrypt'96.

Sch99. B. Schoenmakers. A simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. Crypto'99.

TY. Y. Tsiounis, M. Yung. On the Security of ElGamal based Encryption. *PKC '98*.

VT. E. Verheul, H. van Tilborg. Binding ElGamal: A Fraud-Detectable Alternative to Key-Escrow Proposals. Eurocrypt '97, pages 119–133.
YY98. A. Young, M. Yung. Auto-Recoverable and Auto-Certifiable Cryptosystems. Eurocrypt'98.