Improving Cut-and-Choose in Verifiable Encryption and Fair Exchange Protocols Using Trusted Computing Technology

Stephen R. Tate[1] and Roopa Vishwanathan[2]

[1] Dept. of Computer Science, University of North Carolina at Greensboro Greensboro, NC 27402
srtate@uncg.edu
[2] Dept. of Computer Science and Engineering, University of North Texas Denton, TX 76203
rv0029@unt.edu

Abstract. Cut-and-choose is used in interactive zero-knowledge protocols in which a prover answers a series of random challenges that establish with high probability that the prover is honestly following the defined protocol. In this paper, we examine one such protocol and explore the consequences of replacing the statistical trust gained from cut-and-choose with a level of trust that depends on the use of secure, trusted hardware. As a result, previous interactive protocols with multiple rounds can be improved to non-interactive protocols with computational requirements equivalent to a single round of the original protocol. Surprisingly, we accomplish this goal by using hardware that is not designed for our applications, but rather simply provides a generic operation that we call "certified randomness," which produces a one-way image of a random value along with an encrypted version that is signed by the hardware to indicate that these values are properly produced. It is important to stress that while we use this operation to improve cut-and-choose protocols, the trusted operation does not depend in any way on the particular protocol or even data used in the protocol: it operates only with random data that it generates. This functionality can be achieved with minor extensions to the standard Trusted Platform Modules (TPMs) that are being used in many current systems.

We demonstrate our technique through application to cut-and-choose protocols for verifiable group encryption and optimistic fair exchange. In both cases we can remove or drastically reduce the amount of interaction required, as well as decrease the computational requirements significantly.

1 Introduction

Zero-knowledge proofs were first introduced by Goldwasser et al. [19] and have applications in a wide range of cryptographic protocols that require authentication of one party to another. A zero-knowledge proof is a protocol in which a prover P provides some information I to a verifier V, and then engages in a protocol to convince V that I satisfies some property $Q(I)$ that would be

E. Gudes, J. Vaidya (Eds.): Data and Applications Security 2009, LNCS 5645, pp. 252–267, 2009.

difficult for V to compute on its own. For example, I might be an encrypted value and the property $Q(I)$ refers to a simple property of the corresponding plaintext. Zero-knowledge proofs are used in higher-level protocols such as fair exchange [1], identification protocols [16], and group signatures [4]. Interactive zero-knowledge proof systems often employ a paradigm called "cut-and-choose" in which the prover answers a series of random challenges given by the verifier. For each challenge, the verifier has at most a 50% chance of getting cheated. With sufficiently many challenges, the chances of a dishonest prover answering all of them correctly, and the verifier getting cheated on all of them is negligible. This requires several rounds of communication between the prover and verifier, and increases the communication costs, thereby decreasing the efficiency of the protocol. Several protocols for the verifiable encryption problem use such zero-knowledge proofs, and this is the core problem that we examine in this paper.

Verifiable encryption is a protocol that actively involves two parties, a prover and a verifier, and passively involves one or more additional parties. In its simplest version, with a single trusted third party T, the prover P encrypts a secret value s that is supposed to satisfy some specific property (e.g., a signature on some message) with the public key of the trusted third party PK_T, and sends the encrypted value $E_{PK_T}(s)$ to the verifier V. The protocol is such that V is convinced that the received ciphertext will decrypt to a value s which satisfies the necessary property, but other than this fact V is not able to discover any additional information about the value s. In a typical application, a honest prover will later reveal the secret, and the trusted party is only involved if the protocol does not complete and V needs to recover the secret without the assistance of P. Verifiable encryption has been used to construct solutions for fair exchange [1,2], escrow schemes [29], and signature sharing schemes [17]. In this paper we solve a generalized, more powerful version known as verifiable group encryption, in which there are multiple semi-trusted parties ("recovery agents" or "proxies") and authorized subsets of agents. The secret is encoded such that it can be recovered only by an authorized subset of recovery agents working together.

One of the more important applications of verifiable encryption is the fair exchange problem: Two parties A and B have values that they would like to exchange (e.g., A has a credit card number and B has a downloadable song), and after executing a fair exchange protocol either both parties receive the value they are entitled to, or neither does. Fair exchange is the central problem in various online transactions, and with e-commerce transactions growing at an ever increasing rate with operations such as online credit card transactions, online stock trading, and e-checks becoming more common than ever, a large number of businesses depend on these transactions being executed fairly.

The Trusted Computing Group, an industry consortium of over 100 companies, has developed specifications for a hardware chip called the Trusted Platform Module (TPM) [22]. TPMs have become common in laptops, business-oriented desktops, and tablet PCs, including those by IBM, Dell, Lenovo, HP, and Toshiba, as well as in some server-class systems such as the Dell R300. TPMs are designed to be very cheap (under $5 each), and are intended to be

easily embedded on the motherboard of a standard system. While major CPU and chipset manufacturers have designed additional trusted computing components (e.g., the Intel LaGrande project), in this paper we simply require the TPM chip. Primarily, the TPM is used for measurement of system parameters (BIOS, hardware elements, operating system, software applications, among others) to provide a measured and protected execution environment, which assures the integrity and trustworthiness of the platform. To support trustworthy reporting of these measurements to outside parties, TPMs sign measurements with "Attestation Identity Keys" (AIKs): keys that cannot exist in usable form outside the TPM, are only used inside the TPM to sign values produced inside the TPM chip itself, and are certified by a designated certification authority known as a PrivacyCA which vouches for the fact that the certified key is indeed a legitimate AIK which can be used only internally to a TPM. The exact process of establishing that a key is an internal-use only AIK relies on a chain of trust from the manufacturer of the TPM, and is beyond the scope of this paper to describe — interested readers are referred to the TPM documentation for details [22]. In addition to operations with measurements and AIKs, TPMs provide a variety of other capabilities, and in this paper we use the TPM's ability to generate random numbers (or at least cryptographically secure pseudo-random numbers), and the ability to perform encryption. In this paper we show how to use the capabilities of a TPM to improve upon existing verifiable encryption protocols and protocols (specifically fair exchange) that build upon verifiable encryption.

1.1 Our Contribution

We investigate ways in which limited trusted hardware, such as TPMs, can be used to generate certain non-interactive zero knowledge proofs and thus improve protocols that use these cut-and-choose constructions. Interestingly, we show that a significant benefit can be gained from constructions that we design around two proposed very generic TPM commands: `TPM_MakeRandomSecret` and `TPM_EncryptShare`. These are simple extensions to the current TPM specification that involve generating a (pseudo)-random number and signing a set of values with an AIK. Using these two commands, we can bring down the communication costs and decrease the number of rounds required in certain interactive zero-knowledge proof systems. We have identified two applications of our scheme: fair exchange protocols and verifiable group encryption. Using our TPM-based protocols, we construct a verifiable group encryption scheme and solve an open problem in Asokan *et al.*'s paper [1]: making their protocol non-interactive and bringing down the cost of the *verifiable escrow* operation. Other applications may also benefit from our techniques, including applications such as group signatures and identity escrow schemes.

2 Related Work

This paper builds upon work in both cryptography and in hardware-assisted security, and in this section we review related work in each of these areas.

2.1 Cryptographic Protocols

Cut-and-choose zero-knowledge proofs are used in many cryptographic protocols that involve authentication while maintaining privacy, such as the Fiat-Shamir identification scheme [16,4], non-malleable commitments [18], concurrent zero-knowledge protocols [3], and verifying secret shuffles [20], among other protocols. Although there have been methods proposed to reduce the round complexity of zero-knowledge proofs and the design of non-interactive zero-knowledge proofs, most protocols that use such methods require us to accept a weaker form of security [8]. In this paper we focus on applications of interactive zero-knowledge proofs in verifiable group encryption and a specific application of verifiable encryption: fair exchange of signatures.

We now expand on the description of verifiable encryption given in the previous section to make precise what is meant by the secret value satisfying some property. Specifically, there is a given relation $\mathbf{R}$, the prover and verifier share a public value x, and the secret s is such that $(x, s) \in \mathbf{R}$. For example, x could be a public message, and s could be a digital signature made by the prover on that message — while the verifier is convinced that the prover has indeed signed the message and provided an encrypted version of this signature as $E_{PK_T}(s)$, the verifier cannot retrieve the actual signature until either the prover provides it later or the recovery agent T decrypts the signature for the verifier (e.g., at a specific "release time").

The exact relation $\mathbf{R}$ depends on the application of verifiable encryption. In some protocols, the secret is simply the pre-image of the public value x under some one-way function $f(s) = x$, meaning that $\mathbf{R} = \{(x, f^{-1}(x))\}$ — this is the case in the work by Asokan *et al.* [1], where f is additionally a homomorphism. Camenisch and Damgard [7] consider a generalized problem in which the relation $\mathbf{R}$ is any relation possessing a 3-move proof of knowledge which is an Arthur-Merlin game, which they call a *Σ-protocol*, and includes all of the relations considered by Asokan *et al.*'s earlier work.

In addition to introducing Σ-protocols, Camenisch and Damgard also expand the problem from the verifiable encryption problem studied by Asokan *et al.* (with a single trusted party), to the verifiable group encryption problem (with n semi-trusted recovery agents, or *proxies*), that we described in the previous section. Camenisch and Damgard's solution [7] is a cut-and-choose based method in which the prover generates two sets of n shares of an intermediate secret, and then encrypts these $2n$ shares. The verifier asks the prover to open half of those encryptions, and in doing so can verify that the prover honestly constructed that set of encryptions. Thus if the prover attempts to cheat on one (or both) sets of shares, this will be discovered with probably of at least $1/2$. Camenisch and Damgard then repeat this process multiple times to decrease the chance of being cheated to a negligible probability.

Fair exchange (of digital signatures) is an important application of verifiable encryption. There have been a number of protocols developed for fair exchange of digital signatures, and a survey paper by Ray outlines many of them [30]. The work by Asokan, Shoup, and Waidner [1], which we mentioned above for the

results on verifiable encryption, provides one of the most widely-referenced and efficient optimistic fair exchange protocols. Synchronizing messages between the parties so that the fair exchange properties are met is challenging, but the heart of this protocol is their solution to the verifiable escrow problem. Thus, in case one of the parties cheats, the other (honest) party can get the signature or the promised item from the third party. As is common in many other fair exchange protocols, this protocol employs a cut-and-choose zero-knowledge proof between the prover and verifier.

2.2 Hardware-Assisted Security for Cryptographic Protocols

The idea of using trusted hardware tokens to improve the security of cryptographic protocols can be traced back to the work by Chaum and Pedersen [12], Brands [6], and Cramer [13] in which *observers*, or smart-cards, or tokens act as intermediaries in financial transactions between a user and a bank. This idea was recently studied in a theoretical setting by Katz [26] in which user-constructed hardware tokens are used as part of a protocol for realizing multiple commitment functionality in the universal composability (UC) framework. Independently, Chandran, Goyal and Sahai [11], and Damgard *et al.* [14] improve on Katz's results by making the token independent of the parties using it, resettable, and relying on general assumptions like trapdoor permutations rather than cryptographic assumptions. More recently Moran and Segev [28] have improved upon all of the above results by requiring that only one of the two parties involved needs to build a tamper-proof token as opposed to the previous results which require that both prover and verifier have to generate their own hardware tokens. Our work is clearly related to this line of work on hardware tokens, but rather than using tokens created by a participant in the protocol we trust a hardware manufacturer to produce a trustworthy TPM that provides generic, non-application and non-user-specific functionality. A similar approach using TPMs for a different problem was taken by Gunupudi and Tate [24], who show how a TPM with some slight modifications can be used to act in a way indistinguishable from a random oracle, which can then be used in multi-party protocols.

Sarmenta *et al.* [32] introduced the notion of virtual monotonic counters, and designed two schemes to implement this concept using TPMs. Sarmenta *et al.* then show how virtual monotonic counters can be used to produce several interesting applications such as digital wallets, virtual trusted storage, and offline payment protocols. In addition, they also introduced the concept of *Count-limited objects* (or *clobs*), which are cryptographic keys or other TPM-protected objects that are tied to a virtual monotonic counter, and are limited in the number of times that they can be used.

Using the foundation created by Sarmenta *et al.*, Gunupudi and Tate [25] applied *clobs* to create non-interactive protocols for oblivious transfer, designing a particularly efficient technique for a set of concurrent k-of-n oblivious transfers. In particular, a large set of general oblivious transfers can be performed using just a single clob, resulting in a very efficient protocol for a wide range of applications that use oblivious transfer, including secure function evaluation (SFE).

3 Preliminaries and Protocol Primitives

In this section we describe the TPM operations we propose in this paper. For constructing our protocols, we use the ability of the TPM to generate random (secret) numbers, create signatures using one of the TPM's Attestation Identity Keys (AIKs), and do basic encryption. Recall that an AIK is a signing key that is certified by a PrivacyCA as usable only inside a TPM, so we trust that values signed by an AIK were correctly and honestly produced (assuming an uncompromised TPM).

To support the cryptographic operations required by the TPM specification, TPMs must be able to perform modular arithmetic, and we use these capabilities to perform operations over a particular cyclic group. In particular, let p and q be primes such that $q|p-1$, and let g be a generator for the subgroup of $\mathbf{Z}_p^*$ of order q. These values can be global for all TPMs, although it is advisable that these values be modifiable in case particular primes are discovered to have undesirable properties. The operations we require of the TPM will be modulo p and modulo q arithmetic, and based on other keys used by the TPM and other considerations p might be 1024 or 2048 bits, with q being 160 bits.

3.1 Secret Sharing Schemes

Secret sharing is a fundamental part of our work, so we review the concepts and establish notation in this section. Specifically, we look at *threshold secret sharing*, in which a secret is divided among a group of n members who have decided on a threshold value, say, k. Each member gets a share of the secret, and only groups of at least k members can reconstruct the secret. Shamir's scheme [34] was the first such secret sharing scheme. Other secret sharing variations, using both thresholds and more general notions of access control structures, include those by Feldman [15], Blakley [5] and Krawczyk [27]. Formally, a threshold secret sharing scheme consists of two functions: SHARE and RECOVER such that:

SHARE$(x, k, n) \rightarrow (s_1, s_2, \ldots, s_n)$**:** Splits a secret $x \in \mathbf{Z}_q$ into n shares, with each $s_i \in \mathbf{Z}_q$, so that the secret can be reconstructed from any k shares but not from any fewer than k shares.

RECOVER$(v_1, v_2, \ldots, v_k, k, n) \rightarrow y$**:** If $v_1, v_2, \ldots, v_k$ are k different shares produced by the SHARE operation, then the value y that this produces is the original secret value (x in the SHARE operation).

3.2 Our Protocol Primitives

The following are two proposed TPM commands which we will use in this paper. While not necessary for the most straightforward applications, we give the capability to attach a "condition" to the certified randomness produced by these commands — the condition may be arbitrary, but a fixed-length hash is passed to the first command and then included in each encrypted share by the second command (e.g., if SHA-1 is used, like many other TPM commands, this would

be a 160-bit parameter). This condition is vital for more involved protocols, such as the fair exchange protocol that we describe in Sect. 4.3, where a party uses the condition to commit to certain values when starting the verifiable encryption process.

`TPM_MakeRandomSecret`$(hCond, k, n) \rightarrow (secHandle, r)$: This operation generates a random $r \in \mathbf{Z}_q$ that will be shared in a k-of-n secret-sharing scheme, and provides the value and a handle to an internal protected storage version of the secret. The value $hCond$ is the hash of a "condition" tied to this random secret, as described above.

`TPM_EncryptShare`$(secHandle, AIKHandle, i, PK) \rightarrow$
$Sign_{AIK}(E_{PK}(hCond \parallel s_i), PK, k, n, g^r, hCond)$: Gives a signed, encrypted version of the i-th share of random secret r (referred to by $secHandle$). If this is called with $i = 1, \ldots, n$, the shares that are encrypted, $s_1, \ldots, s_n$ should be the shares output by $\textsc{Share}(r, k, n)$ for some secret sharing scheme.

Since TPMs support basic modular arithmetic for encryption operations, these operations are relatively easy to implement efficiently using Shamir's shecret sharing scheme [34]. In some applications, such as the optimistic fair exchange protocol that we describe later, we don't need secret sharing at all. Instead, we simply need to be able to encrypt the secret random value using the public key of one or more trusted parties so that the secret can be recovered later if necessary. This is a degenerate "1-of-n" case of the generic operation above, but we describe it here separately as this might be the most useful form of the certified randomness operations.

`TPM_MakeRandomSecret`$(hCond) \rightarrow (secHandle, r)$: Random $r \in \mathbf{Z}_q$ is selected, stored internally with $secHandle$ to refer to it, and returned to the user. $hCond$ is the hashed condition, as before.

`TPM_EncryptShare`$(secHandle, AIKHandle, PK) \rightarrow$
$Sign_{AIK}(E_{PK}(hCond \parallel r), PK, g^r, hCond)$: The previously generated r is bundled with the condition $hCond$ and encrypted for a trusted party whose public key is denoted by PK, and signed by the Attestation Identity Key (AIK).

4 Our TPM-Based Verifiable Group Encryption Protocol

In this section we show how the TPM operations defined above can be used to implement a form of verifiable group encryption, a problem defined by Camenisch and Damgard [7]. In verifiable group encryption, there is a sender (prover), a receiver (verifier), and n semi-trusted parties ("proxies"). The sender has a public value x that it claims satisfies some property, which can be verified using a secret witness s known to the sender. The sender wants to send x along with additional information to the receiver such that the receiver (a) has assurance that it can recover the secret s if it has the cooperation of an authorized subset of the proxies and (b) without the cooperation of an authorized subset of proxies the receiver

gets no information about s. There are several ways to define an "authorized subset" of proxies, but the most generic way is to use a monotone access structure Γ, which is a set of subsets of authorized proxies that is monotone (so that if $A \in \Gamma$ and $A \subseteq B$ then $B \in \Gamma$).

We next provide a precise and formal definition of this problem. While Camenisch and Damgard formally defined (non-group) Verifiable Encryption and then informally described the group encryption problem [7], they did not formally define the Verifiable Group Encryption problem. Therefore, while our definition is based on their work, the definition given here is new.

While expressed formally, the properties in the definition have simple intuitive descriptions: The *Completeness* property states that if the prover and verifier are honest, the verifier always accepts. The *Validity* property says that no dishonest prover can trick a verifier into accepting something that will not allow recovery of a valid witness s if the verifier uses a set of proxies that are included in the access control structure Γ. The *Computational Zero Knowledge* property states that no dishonest verifier can recover any information about a valid witness s, and this is true even if the verifier uses an arbitrary set of proxies that is not in the access control structure.

Definition 1 (Generic Verifiable Group Encryption). *Let* $\mathbf{R}$ *be a relation and define language* $L_{\mathbf{R}}$ *by* $L_{\mathbf{R}} = \{x \,|\, \exists s : (x, s) \in \mathbf{R}\}$. *A* Verifiable Group Encryption Scheme *for relation* $\mathbf{R}$ *consists of a two-party protocol* (P, V) *and a recovery algorithm* R. *Assume we have a set of* n *proxies with encryption/decryption algorithms denoted* (E_i, D_i) *for* $i = 1, \ldots, n$, *and denote this set with notation* $(\mathcal{E}, \mathcal{D}) = ((E_1, E_2, \ldots, E_n), (D_1, D_2, \ldots, D_n))$. *Let* $V_P(\mathcal{E}, x, \Gamma, \lambda)$ *denote the output of the verifier when interacting with* P *on input* $(\mathcal{E}, x, \Gamma, \lambda)$, *where* Γ *is a monotone access structure over* $\mathcal{D}$ *and* λ *is a security parameter. Let* $\perp$ *denote the null set. The following properties must hold:*

1. *Completeness: If* P *and* V *are honest, then for all sets of proxies* $(\mathcal{E}, \mathcal{D})$ *and all* $x \in L_{\mathbf{R}}$,
$$V_P(\mathcal{E}, x, \Gamma, \lambda) \neq \perp \ .$$
2. *Validity/Soundness: For all polynomial time* $\tilde{P}$, *all* $(\mathcal{E}, \mathcal{D})$, *all* Γ, *all* $\tilde{\mathcal{D}} \in \Gamma$, *all positive polynomials* $p(\cdot)$, *and all sufficiently large* λ, *if* $\alpha = V_{\tilde{P}}(\mathcal{E}, x, \Gamma, \lambda)$ *and* $\alpha \neq \perp$ *then*
$$Prob[(x, R(\tilde{\mathcal{D}}, \alpha)) \notin \mathbf{R}] < 1/p(\lambda) \ .$$
3. *Computational Zero Knowledge: For any polynomial time* $\tilde{V}$, *which is a verifier that takes a subset of decryption proxies* $\tilde{\mathcal{D}} \subseteq \mathcal{D}$ *in addition to the regular verifier parameters, there exists an expected polynomial time simulator* $S_{\tilde{V}}$ *with black-box access to* $\tilde{V}$ *such that for all polynomial time distinguishers* A, *all positive polynomials* p, *all* $x \in L_{\mathbf{R}}$, *and all sufficiently large* λ, *if* $\tilde{D} \notin \Gamma$ *we have*
$$Prob[A(\mathcal{E}, x, \alpha_i) = i : \alpha_0 = S_{\tilde{V}}(\mathcal{E}, x, \lambda);\ \alpha_1 = \tilde{V}_P(\mathcal{E}, \tilde{D}, x, \lambda);\ i \in_R \{0, 1\}]$$
$$< \frac{1}{2} + \frac{1}{p(\lambda)} \ .$$

Definition 1 is a generic form of verifiable group encryption, but in order to produce algorithms that can be put into fixed trusted hardware, we restrict two general parameters in the above definition — we fix the relation $\mathbf{R}$ to be a specific relation based on the discrete log problem, and we restrict the access control structure Γ to be a threshold structure so that we can use simple threshold secret sharing schemes. Specifically, we make the following two restrictions:

1. $\mathbf{R} \subseteq (\mathbf{Z}_p, \mathbf{Z}_q)$ is defined so that $\mathbf{R} = \{(g^s, s) \,|\, s \in \mathbf{Z}_q\}$. In other words, in any pair $(x, s) \in \mathbf{R}$, the secret s is the discrete log of x.
2. $\Gamma = \{\tilde{\mathcal{D}} \,|\, \tilde{\mathcal{D}} \subseteq \mathcal{D} \text{ and } |\tilde{\mathcal{D}}| \geq k\}$

Since we will be using trusted platforms to implement verifiable group encryption, we need to establish the security properties of a trusted platform so that we can reason about the security of our implementation. This assumption is nothing more than an explicit description of what it means for a TPM to be secure in the obvious sense, and reflects the requirements given in the TPM Protection Profile [21]. While no hardware can be perfectly protected, currently manufactured TPMs appear to have a high level of tamper resistance with regard to the protection of secrets, and we believe that our security assumption is realistic.

Definition 2. *The* Trusted Platform Security Assumption *is the assumption that the system containing a TPM satisfies the following properties:*

1. Tamper-resistant hardware: *It is infeasible to extract secrets stored in protected locations in the TPM.*
2. Secure Encryption: *The public-key encryption algorithm used by the TPM is CCA-secure.*
3. Secure Signatures: *The digital signature algorithm used by the TPM is existentially unforgeable under adaptive chosen message attacks.*
4. Trustworthy PrivacyCA: *Only valid TPM-bound keys are certified by a trusted PrivacyCA.*

Assuming we have a trusted platform that provides the operations defined in Sect. 3.2 and satisfies the Trusted Platform Security Assumption, we define the following protocol for the verifiable group encryption problem — since our protocol is non-interactive, we define it as a pair of algorithms, one for the sender which produces the verifiably-encrypted secret, and one for the receiver which verifies that the values produced by the sender are properly formed.

In the following algorithms, $PK_1, \ldots, PK_n$ denote the public encryption keys of the n proxies, so the n-tuple $(PK_1, \ldots, PK_n)$ is the realization of the abstract set of encryption routines $\mathcal{E}$ given in Definition 1. Furthermore, since the abstract access structure Γ is restricted to be a threshold structure, it is fully specified by the pair (k, n), which represents a "k-of-n" threshold structure. Finally, we assume that the AIK used in this protocol is loaded into the TPM before VESender is called and is referenced by TPM handle $AIKH$, and we have a certificate $Cert(AIK)$ for this AIK signed by a trusted PrivacyCA. The condition $hCond$ associated with this escrow is the same as described in Sect. 3.2, and can be used as needed by applications.

Algorithm. VESENDER$((PK_1, \ldots, PK_n), s, (k, n), hCond, \lambda)$
$x \leftarrow g^s$
$(secHandle, r) \leftarrow$ `TPM_MakeRandomSecret`$(hCond, k, n)$
$d \leftarrow s + r \bmod q$
$t \leftarrow g^r \bmod q$
for $i \in 1, \ldots, n$ **do** $C_i \leftarrow$ `TPM_EncryptShare`$(secHandle, AIKH, i, PK_i)$
return $\mathcal{B} = \langle d, x, t, C_1, C_2, \ldots, C_n, Cert(AIK)\rangle$

Algorithm. VERECVERIFY$(\mathcal{B} = \langle d, x, t, C_1, C_2, \ldots, C_n, Cert(AIK)\rangle, hCond)$
if $Cert(AIK)$ is not certified by a trusted PrivacyCA **then return** $\perp$
if $C_1, \ldots, C_n$ are not tuples $\langle c_i, PK_i, k, n, t, h\rangle$ signed by AIK **then return** $\perp$
if $\langle k, n, t, h\rangle$ are not identical in all tuples with $h = hCond$ **then return** $\perp$
if $g^d \neq xt$ **then return** $\perp$
return $\alpha = \mathcal{B}$

If it is necessary to use these encryptions to recover the secret s, the following algorithm can be used:

Algorithm. VERECOVER$(\alpha = \langle d, x, t, C_1, C_2, \ldots, C_n, Cert(AIK)\rangle, hCond)$
Select k C_i's: $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$, and use proxies to decrypt each $hCond_{i_k} \parallel s_{i_k}$
if any $hCond_{i_k}$ does not match parameter $hCond$ **then return** $\perp$
$r \leftarrow$RECOVER$(s_{i_1}, s_{i_2}, \ldots, s_{i_k}, k, n)$
$s' \leftarrow (d - r) \bmod q$
return s'

Since the TPM guarantees that the recovered r is such that $t = g^r$, and we have verified that $g^d = xt$, we have $g^d = xg^r$, so s' is such that $g^{s'} = g^{d-r} = x$. Since s is the unique value in $0, \ldots, q-1$ such that $g^s = x$, we must have $s' = s$, and we have recovered the original secret s. Below we demonstrate a useful application of this scheme.

4.1 Application to Verifiable Group Encryption of Signatures

In this section, we show how to apply our verifiable group encryption protocol to the problem of verifiably encrypting shares of a valid digital signature. In this problem there is a publicly-known message m, and we would like for a party with public key y to sign this message. The signature is not to be revealed immediately, but needs to be committed to and escrowed in such a way that an appropriate set of trusted parties can recover the signature if necessary. We would like to verifiably encrypt the signature so that the receiver has assurance that the ciphertexts which it receives (and which are unintelligible to it) are in fact shares of the requested signature.

Our verifiable encryption protocol can be used with many signature schemes, including GQ [23], Fiat-Shamir [16], and RSA [31], but for concreteness we instantiate it here with Schnorr signatures [33]. Techniques for using other signature schemes are similar to the techniques described in Section 4 of Asokan *et al.* [1]. Note that the verifiable encryption protocol is independent of the signature scheme used, and the security of the verifiable encryption is in no way related to the security of the underlying signature scheme. The Schnorr signature scheme is briefly described below:

SETUP OF SYSTEM PARAMETERS: There is a prime modulus p with a generator g of a subgroup of $\mathbf{Z}_p^*$ of prime order q, where q is also prime. A random value $\zeta \in \mathbf{Z}_q$ is the private key of the signer, Alice, and her public key is $y = g^\zeta$. Let Bob be the verifier.

SCHNORRSIGN(m, ζ): To create her signature, Alice picks a random $r \in \mathbf{Z}_q$ and computes $u = g^r$, $c = h(u \parallel m)$, and $z = r + \zeta c \bmod q$, where h is a cryptographic hash function. Her signature is then (c, z).

SCHNORRVERIFY$((c, z), m, y)$: Bob, who knows Alice's public key y, checks if $h(g^z y^{-c} \parallel m) = c$, and accepts if it is true, rejects otherwise.

Next we show the sender and receiver portions of our verifiable group encryption of a Schnorr signature. Since the Schnorr signature is a pair (c, z), where c is random (in the random oracle model), we transmit c in the clear and use the verifiable encryption for only the second component z.

Algorithm. SIGSENDER$((PK_1, \ldots, PK_n), m, (k, n), \zeta, hCond)$
$(c, z) \leftarrow$SCHNORRSIGN(m, ζ)
$\mathcal{B} \leftarrow$VESENDER$((PK_1, \ldots, PK_n), z, (k, n), hCond, \lambda)$
return $\langle (c, z), \mathcal{B} \rangle$

Algorithm. SIGVERIFIER$(y, m, c, \mathcal{B} = \langle d, x, t, C_1, C_2, \ldots, C_n, Cert(AIK) \rangle, hCond)$
if y is not an acceptable public key **then return** $\perp$
if $h(xy^{-c} \parallel m) \neq c$ **then return** $\perp$
return VERECVERIFY$(\mathcal{B}, hCond)$

If it is necessary to recover the signature from the encrypted shares, we first do a recovery from the verifiable group encryption to recover z. Since SIGVERIFIER has verified that $h(xy^{-c} \parallel m) = c$, and the recovered z is such that $x = g^z$, we have $h(g^z y^{-c} \parallel m) = c$, which is exactly the property that must be verified in SCHNORRVERIFY. Therefore, (c, z) is a valid Schnorr signature on message m.

Algorithm. SIGRECOVER$(c, \mathcal{B}, hCond)$
$z \leftarrow$ VERECOVER$(\mathcal{B}, hCond)$
if $z = \perp$ **then return** $\perp$
return (c, z)

Our building blocks for this protocol, VESENDER and VERECVERIFY, provide a secure verifiable group encryption scheme, reflected in the following theorem.

Theorem 1. *Let* $\mathbf{R}$ *be a relation such that* $\mathbf{R} \subseteq (\mathbf{Z}_p, \mathbf{Z}_q)$ *is defined so that* $\mathbf{R} = \{(g^z, z) \mid z \in \mathbf{Z}_q\}$. *The protocol outlined above, when the prover uses a system that satisfies the Trusted Platform Security Assumption, is a secure verifiable group encryption scheme for* $\mathbf{R}$.

Proof: Due to space limitations we have removed this proof from the conference paper. It can be found in the full version [35].

4.2 Comparison to Non-trusted Platform Protocols

We now briefly compare the costs of our protocol to two verifiable encryption protocols: that of Camenisch and Damgard [7] (the "CD" protocol), and Camenisch and Shoup [9] (the "CS" protocol), in Table 1. Note that the CS protocol is described only for verifiable encryption of a single secret (not verifiable group encryption); in the table, we estimate the values for k shares.

Although we cannot fully describe these protocols here due to lack of space, we note that our full protocol is roughly comparable to one round of the CD protocol and is completely non-interactive as compared to the CS protocol. The CD protocol must be repeated in order to build trust: a dishonest prover can get away with cheating on a single iteration of the secret-sharing phase with probability $1/2$, and this probability is reduced to $1/2^R$ by repeating the protocol R times. Using 30 rounds means the probability of the prover cheating without detection is about one in a billion, and a very high degree of assurance can be obtained by repeating the protocol 80-100 times.

The CS protocol improves on the CD protocol, but is not completely non-interactive, although it reduces the amount of interaction from that required by the CD protocol. In addition to providing improved efficiency, our protocol is completely non-interactive: the prover computes some values, sends them off, and then is no longer involved. In an e-mail setting (send and forget), this is a vital property that our protocol achieves but earlier solutions do not.

4.3 Fair Exchange of Digital Signatures

Fair exchange is an important application of the verifiable group encryption protocol outlined above, where additional operations are included to synchronize the actions to ensure that the fair exchange property is satisfied. One of the open problems left by Asokan *et al.* [1] was to come up with a way to eliminate the expensive verifiable escrow operation and make their protocol non-interactive. By using TPMs, we have shown that we can indeed make the protocol non-interactive, greatly improving efficiency.

Table 1. Comparison of the cost of the CD [7] and CS [9] protocols to our TPM-based solution for n proxies

CD protocol		CS protocol	Our Solution
Interactive		Interactive	Non-interactive
Cost for *One Round* (of 30-100)		Cost for Full Solution	Cost for Full Solution
Prover:	2 n-party SHAREs	1 n-party SHARE	1 n-party SHARE
	$2n$ encryptions	n encr.	n encr.
		n signatures	n sig.
Verifier:	n encrytions	n sig. verifies	$n+1$ sig. verifies
	1 or 2 mod powerings	1 to 4 mod pow.	1 mod pow.
	0 or 3 mod multiplies	1 to 4 mod mult.	1 mod mult.
Recovery:	k decryptions	k decr.	k decr.
	1 secret sharing RECOVER	1 sec. shar. RECOVER	1 sec. shar. RECOVER

In fair exchange of signatures, two parties have messages that they have pledged to sign, and at the end of the protocol either both have received the other's signature, or neither has. For concreteness, we assume Schnorr signatures (as we used in our SigSender function), but this can easily be adapted to any signature scheme that has a secure reduction scheme using the homomorphism $\theta(x) = g^x$. At the beginning of the protocol, parties A and B have agreed to sign messages m_A and m_B, respectively, and along with the publicly-known messages both parties know each other's signature public keys y_A and y_B (corresponding private keys ζ_A and ζ_B are known only to A and B, respectively). Asokan *et al.*'s protocol is an "optimistic fair exchange protocol," meaning that the trusted third party is not involved unless there is a dispute, but the third party does require a single, consistent database of tuples to keep track of any requests that have been made of it. The generality of our solutions for verifiable encryption in this paper, using multiple trusted parties and a threshold scheme, causes complications for the fair exchange problem — unless a single globally-consistent database is maintained, it might be possible for a party to cheat the controls that the trusted party is supposed to enforce. While we could potentially do this with some distributed database of tuples, we simplify the problem back to a single trusted party in this section. The trusted party has public encryption key PK, which is known to all parties.

Below we give our modification of Asokan *et al.*'s fair exchange protocol, which has our construct incorporated. In addition to inserting our signature escrow functions SigSender, SigVerifier, and SigRecover, a few changes come from the fact that we are using a specific signature scheme rather than a generic homomorphic reduction scheme.

1. A chooses $r \in \mathsf{Domain}(f)$ at random and computes $v = f(r)$. A then computes its signature σ_A, encrypts it using a regular escrow scheme with condition (v, m_B, y_B), giving an escrow α which it sends along with v to B.
2. B receives v and α from A, and computes $hCond = h(\langle v, \alpha, m_A, y_A \rangle)$, then $\langle (c_B, z_B), \beta \rangle = \text{SigSender}(PK, m_B, \zeta_B, hCond)$. B's signature on m_B is then $\sigma_B = (c_B, z_B)$. B sends c_B and β to A.
3. A receives c_B and β from B, computes it's own copy of $hCond$ and checks that β is a proper signature escrow using $\text{SigVerifier}(y_B, m_B, c_B, \beta, hCond)$ — if this does not verify, A calls $\text{A-Abort}(r, m_B, y_B)$ and quits the protocol (perhaps having received B's signature σ_B); otherwise, A creates signature σ_A for message m_A and sends σ_A to B.
4. B receives σ_A from A and verifies that this is a valid signature on m_A. If this is a valid signature, then B sends signature σ_B to A; otherwise, B calls $\text{B-Resolve}(v, \alpha, m_A, y_A, m_B, y_B, \sigma_B)$ which either returns σ_A (if α is a valid escrow) or an error message.
5. A receives σ_B from B and verifies that this is a valid signature on m_B. If this is a valid signature, then the protocol halts, with both parties having received valid signatures; otherwise, A calls $\text{A-Resolve}(r, \alpha, \beta, c_B, m_B, y_B, m_A, y_A)$ to get σ_B.

Functions A-Abort, A-Resolve, and B-Resolve are executed (atomically) by the trusted party, and are straightforward adaptations of the functions designed by Asokan *et al.* [1] following changes we made in the base exchange protocol above. The details of these functions are provided in the full paper [35].

The fairness of this protocol is established following reasoning similar to that in Asokan *et al.* [1], using Theorem 1 in this paper for the security of our verifiable encryption scheme, resulting in the following theorem. The proof of this theorem is a simple modification to the proof in Asokan *et al.* [1], so is not given here.

Theorem 2. *Given parties A and B, in which B has access to a TPM that satisfies the Trusted Platform Security Assumption, the protocol described above is a secure optimistic fair exchange protocol.*

The most expensive operation in Asokan *et al.*'s protocol is the verifiable escrow operation which makes the cost of the protocol grow with R, where R is the number of rounds between prover and verifier. Using our functions in place of the verifiable escrow, we only need a single round, reducing the computational cost, and perhaps more importantly makes that part of the fair exchange protocol non-interactive.

5 Conclusion and Future Work

We have presented a generic subroutine-like TPM-based construct that we used to create algorithms that replace cut-and-choose protocols in some interactive zero-knowledge proofs, making that part of the protocol non-interactive and more computationally efficient. In the process we have provided an efficient protocol for verifiable group encryption and improved the efficiency of the protocol for fair exchange of signatures due to Asokan *et al.* [1]. Our protocols are generic and independent of the signature scheme being used.

An intestesting open problem is whether other cut-and-choose protocols can also be replaced with the help of the TPM-based techniques developed in this paper, and we are exploring the possibility that our construct could be used to improve other applications. Another interesting direction to pursue would be looking at whether the security our TPM-based techniques can be reasoned about in Canetti's strong universally composable model of security [10]. We believe that this is indeed possible, and could provide results quite similar to Katz's results but using our standard hardware components rather than custom-designed hardware tokens.

References

1. Asokan, N., Shoup, V., Waidner, M.: Optimistic Fair Exchange of Digital Signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
2. Bao, F., Deng, R., Mao, W.: Efficient and Practical Fair Exchange Protocols with an Off-line TTP. In: Proceedings of the 19th IEEE Symposium on Security and Privacy, pp. 77–85 (1998)

3. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent Non-Malleable Zero Knowledge. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 345–354 (2006)
4. Bellare, M., Palacio, A.: GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and concurrent attacks. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 162–177. Springer, Heidelberg (2002)
5. Blakley, G.: Safeguarding Cryptographic Keys. In: AFIPS National Computer Conference, pp. 313–317 (1979)
6. Brands, S.: Untraceable Off-line Cash in Wallets with Observers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994)
7. Camenisch, J., Damgard, I.: Verifiable Encryption, Group Encryption, and their Applications to Separable Group Signatures and Signature Sharing Schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (2000)
8. Camenisch, J., Lysyanskaya, A.: An identity escrow scheme with appointed verifiers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 388–407. Springer, Heidelberg (2001)
9. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
10. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: IEEE Symposium on Foundations of Computer Science, pp. 136–145 (2001)
11. Chandran, N., Goyal, V., Sahai, A.: New Constructions for UC Secure Computation using Tamper Proof Hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg (2008)
12. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers (extended abstract). In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
13. Cramer, R., Pedersen, T.J.: Improved Privcy in Wallets with Observers (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 329–343. Springer, Heidelberg (1994)
14. Damgard, I., Nielsen, J.B., Wichs, D.: Isolated Proofs of Knowledge and Isolated Zero Knowledge. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 509–526. Springer, Heidelberg (2008)
15. Feldman, P.: A Practical Scheme for Non-interactive Verifiable Secret Sharing. In: FOCS 1987, pp. 427–437. IEEE Computer Society, Los Alamitos (1987)
16. Fiat, A., Shamir, A.: How to prove to yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
17. Franklin, M., Reiter, M.: Verifiable Signature Sharing. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 50–63. Springer, Heidelberg (1995)
18. Gennaro, R.: Multi-trapdoor Commitments and their Applications to Proofs of Knowledge Secure under Concurrent Man-in-the-Middle Attacks (2004)
19. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. SIAM Jounal of Computing, 186–208 (1989)
20. Groth, J.: A verifiable Secret Shuffle of Homomorphic Encryptions. In: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography, pp. 145–160 (2003)

21. Trusted Computing Group. Protection profile — PC client specific trusted platform module, https://www.trustedcomputinggroup.org/specs/TPM/ (TPM Family 1.2; Level 2)
22. Trusted Computing Group. Trusted Platform Module Specifications – Parts 1–3, https://www.trustedcomputinggroup.org/specs/TPM/
23. Guillou, L., Quisquater, J.: A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
24. Gunupudi, V., Tate, S.R.: Random Oracle Instantiation of Distributed Protocols using Trusted Platform Modules. In: 21st International Conference on Advanced Information Networking Applications, pp. 463–469 (2007)
25. Gunupudi, V., Tate, S.R.: Generalized Non Interactive Oblivious Transfer using Count-Limited Objects with Applications to Secure Mobile Agents. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 98–112. Springer, Heidelberg (2008)
26. Katz, J.: Universally Composable Multi Party Computation using Tamper-proof Hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
27. Krawczyk, H.: Secret sharing made short. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 136–146. Springer, Heidelberg (1994)
28. Moran, T., Segev, G.: David and Goliath Commitments: UC Computation for Asymmetric Parties using Tamper-proof Hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 527–544. Springer, Heidelberg (2008)
29. Poupard, G., Stern, J.: Fair Encryption of RSA Keys. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 172–190. Springer, Heidelberg (2000)
30. Ray, I., Ray, I.: Fair Exchange in E-Commerce. In: Proceedings of ACM SIGEcom Exchange, pp. 9–17 (2002)
31. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 120–126 (1978)
32. Sarmenta, L.F.G., van Dijk, M., O'Donnell, C.W., Rhodes, J., Devadas, S.: Virtual Monotonic Counters and Count-Limited Objects using a TPM without a Trusted OS. In: STC 2006 (2006)
33. Schnorr, C.: Efficient Signature Generation by Smart Cards. Journal of Cryptology, 161–174 (1991)
34. Shamir, A.: How to share a secret. Communications of the ACM, 612–623 (1979)
35. Tate, S.R., Vishwanathan, R.: Improving cut-and-choose in verifiable encryption and fair exchange protocols using trusted computing technology, http://eprint.iacr.org/