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

Stephen R. Tate
Department of Computer Science
Univ. of North Carolina at Greensboro
Greensboro, NC 27402

srtate@uncg.edu

Roopa Vishwanathan
Dept. of Computer Science and Engineering
University of North Texas
Denton, TX 76203

rv0029@unt.edu

August 30, 2009

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.

*An abbreviated, preliminary version of this work appeared in the *Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'09)*.

1 Introduction

In this paper, we explore the question of how much power trusted hardware adds in designing cyrptographic protocols, looking specifically at certain zero-knowledge proofs. Zero-knowledge proofs were first introduced by Goldwasser et al. [21] 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 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 [17], 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 [35], signature sharing schemes [19] and publicly verifiable secret sharing [40]. 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. Publicly verifiable secret sharing (PVSS) introduced by Stadler [40] operates in the same way. In PVSS, there is a "dealer" who makes shares of a secret value under the keys of n participants $P_1, \cdots, P_n$ and distributes the shares such that any coalition of k participants can recover the secret. It also has the property that anybody can verify that the secret shares are correctly distributed. This is just a restatement of the verifiable group encryption problem.

The problem we address in this paper is actually a cross between this verifiable group encryption problem and the verifiable escrow problem of Asokan *et al.* [1]. The difference between verifiable encryption and verifiable escrow is that verifiable escrow attaches an arbitrary "condition" (sometimes called a label), given as a binary string, to an encryption such that the same condition must be supplied to the trusted third party when requesting a decryption. As described by Asokan *et al.*, verifiable encryption and verifiable escrow are equivalent in the sense that a solution to either one can be used in a straightforward way to create a solution for the other. Our solutions all take a condition, so are escrow schemes, but parties can use a simple constant condition (such as a string of zeroes) can be used if the condition is not needed.

One of the more important applications of verifiable encryption is the fair exchange problem: Two par-

ties 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. Not just financial transactions, but medical data and software too are exchanged over the Internet. In light of this, ensuring fair exchange in an online transaction becomes increasingly important. There are a number of protocols that are based on fair exchange, each addressing a different kind of transaction, including credit card transactions and e-payment protocols [31, 32, 33], and non-repudiation in exchange of digital signatures [41]. In Section 2 we briefly review some of the work in this area.

In this paper we consider the use of security hardware to replace the cut-and-choose proofs used in some protocols for verifiable escrow/encryption and fair exchange, with hardware similar to that specified by the Trusted Computing Group (TCG). The TCG is an industry consortium of over 100 companies that has developed specifications for various security technologies, including a hardware chip called the Trusted Platform Module (TPM) [24]. 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 [24]. In addition to operations with measurements and AIKs, TPMs provide a variety of other capabilities, including the ability to generate random numbers (or at least cryptographically secure pseudo-random numbers), support for common cryptographic algorithms including RSA (for encryption and digital signatures), SHA-1 (for generating hashes), and HMAC (hashed message authentication code), as well as more specialized operations such as a privacy-oriented authentication technique known as direct anonymous attestation [7]. In addition to specifying functionality, the Trusted Computing Group has defined a Common Criteria protection profile [23] so that devices can be validated by third parties in order to increase trust in the device.

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. We note that verifiable secret sharing uses the same idea as verifiable group encryption and our protocol can be seen as an improvement to verifiable secret sharing as well.

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 escrow/encryption. Using our TPM-based protocols, we construct a verifiable group escrow 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. We note that Camenisch and Shoup have also designed a verifiable encryption protocol that avoids cut-and-choose [9], but there are certain differences and advantages to our solution, which we expand on below in "Previous Work" and further in Section 4.3. In addition to this, the algorithms proposed for verifiable group encryption in this paper can also be used in publicly verrifiable secret sharing (PVSS) schemes since both are essentially the same problem. Even beyond direct comparisons as solutions to the verifiable encryption problem, our motivation is to study how trusted hardware can benefit cut-and-choose, and as a general construction 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 [17, 4], non-malleable commitments [20], concurrent zero-knowledge protocols [3], and verifying secret shuffles [22], among other protocols. While cut-and-choose is a powerful technique, such protocols tend to be inefficient and require significant interaction, so a lot of work has been done in removing cut-and-choose from specific protocols. In this paper we investigate how cut-and-choose can be removed through the use of trusted hardware devices, with our main focus being on 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 prover also knows a secret s such that $(x, s) \in \mathbf{R}$. The prover and verifier agree on a trusted third party with public encryption key PK_T, and the prover sends the verifier ciphertext $E_{PK_T}(s)$ and proves in zero-knowledge that the ciphertext is really a properly-encrypted valid secret. 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 homomor-

phism. Camenisch and Damgard [8] 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. Camenisch and Shoup [9] consider a restricted version where the secret s is the discrete log of the public value x (this is also the relation we consider in this paper).

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 [8] 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.

Camenisch and Shoup developed a solution to the verifiable encryption problem, where the relation is restricted to be the discrete log, in which they avoid cut-and-choose and thereby significantly improve upon the efficiency of the previous solutions [9]. In order to accomplish this, they design a new encryption scheme whose security depends on Paillier's decision composite residuosity assumption, which they then use in constructing a verifiable encryption protocol. While this is a significant improvement, it relies on a custom encryption scheme rather than the well-studied standard schemes used by previous protocols. Furthermore, it does not remove all interaction: the verifiable encryption scheme still requires a three-round protocol, so is not suitable for "send-and-forget" schemes like emailing a verifiable encryption. Finally, there is no clear way to convert their verifiable encryption scheme into a verifiable group encryption scheme, as addressed by Camenisch and Damgard — in particular, the same modulus would not be useable by multiple independent third parties as it is in schemes based on the standard discrete log problem over a prime-order cyclic group.

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 [36]. Fair exchange protocols can be broadly classified into three categories: Protocols requiring a third party that is always online, protocols that do not need a third party, but which rely on "gradual fair exchange," and protocols that work with an offline third party. Offline trusted third party-based protocols are also known as *Optimistic Fair Exchange* protocols, where the third party is not actively involved in the protocol, but whenever one of the two parties cheats, the honest party can go to the third party, who will resolve the transaction. 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. In this protocol, the parties first exchange encrypted versions of their signatures and then in the second step exchange plaintext signatures. In case either of them refuse to reveal their plaintext signature, the other party can take the encrypted version to the third party who will resolve the transaction. The trusted party ensures that the exchange is fair, and there are built-in constructs to deal with the situation where one party unfairly accuses the other of cheating and asks the third party to intervene. As is common in many other fair exchange protocols, this protocol employs a cut-and-choose zero-knowledge proof between the prover and verifier.

Another application of verifiable group encryption is publicly verifiable secret sharing (PVSS) as used

by Stadler [40]. Although Stadler uses different techniques (double discrete logarithms) to construct a PVSS scheme; in principle, PVSS is nothing but verifiably sharing a secret value among a group of parties such that only an authorized subset of them can recover the secret. Our verifiable group encryption protocol solves exactly this problem and hence is applicable to PVSS protocols too.

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 [14] 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 [28] in which user-constructed hardware tokens are used as part of a protocol for realizing multiple commitment functionality in the universal composability (UC) framework. An important contribution of Katz's paper is that to reduce or possibly eliminate the need for trusted third parties in the UC model, Katz presents a new setup assumption using tamper-proof hardware tokens. Similar to our work, the hardware tokens provide a fairly generic functionality, but to remove the necessity of trust in a third party Katz assumes that each party is responsible for the complete construction of its hardware token.

Independently, Chandran, Goyal and Sahai [11], and Damgard *et al.* [15] improve on Katz's results by making the token independent of the parties using it, resettable, and relying on general assumptions like trapdoor permutations. More recently Moran and Segev [34] 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 parties 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. In other work looking at standard hardware, Hazay and Lindell look at secure protocol design using standard smartcards [27], with solutions for oblivious set intersection and various database search operations.

A similar approach using TPMs for a different problem was taken by Gunupudi and Tate [25], 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. This is accomplished by using a Certifiable Migratable Key (CMK), which is supported by current TPMs, as a seed for an HMAC function (which is also supported by current TPMs). The resulting functionality is called a "TPM-Oracle," and Gunupudi and Tate proved that any protocol secure in the random oracle model is also secure when using a TPM-Oracle.

Sarmenta *et al.* [37] 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 [26] used *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). In addition, Gunupudi and Tate give an application of this protocol to secure mobile agents, removing interaction requirements that were present in all previous secure agent protocols.

2.3 Our Work in Relation to Previous Work

As noted above, a TPM is a practical way of providing functionality similar to the hardware token considered by Katz [28], although the operations implemented must be carefully chosen since a TPM may not be as powerful as the abstract black-box token used in Katz's work. Arguably, in the real world today, hardware devices should be classified as *tamper-resistent* rather than *tamper-proof*, as it is hard to imagine any real-world physical token that is tamper-proof in the strictest sense, meaning that it can thwart all physical attacks. However, TPMs are designed to provide a high level of protection to their internal protected data, and TPMs have the capability of wiping stored data if an attack is detected, increasing the trust that a user can have in both its own TPM and remote TPMs.

Previous work [25, 26, 37] has shown that minor and realistic extensions to the TPM specification are worth considering, if they result in significant performance improvements in the protocols in which they are used. Katz's paper and the subsequent papers on tamper-proof hardware [11, 15, 28] all demonstrate the power of hardware-assisted security. The TPM commands we propose in this paper provide a practical way in which users could directly use these primitives in their applications.

As a solution to the verifiable encryption problem, our solution has several advantages over previous solutions, albeit with the requirement of hardware additions. While Camenisch and Shoup [9] present an efficient solution that does not require cut-and-choose, their solution required the development of a new encryption scheme, while ours can use standard well-studied encryption methods. Previous work by Asokan *et al.* and Camenisch and Damgard also used standard encryption, but required cut-and-choose. There are additionally some other advantages of our solution when compared to the Camenisch/Shoup solution, including working over a smaller modulus and more compact ciphertexts. Furthermore, our solution is the only one that is non-interactive: the cut-and-choose solutions require a large amount of interaction, and the Camenisch and Shoup solution is a three-round protocol. As a result, our solutions is the only one appropriate for an e-mail like setting, where the ciphertext and zero-knowledge proof can be sent off and require no further input from the prover. Finally, like the Camenisch/Damgard protocol for verifiable group encryption, our solution can also handle general settings with sets of trusted third parties, and it is not clear that the Camenisch/Shoup protocol can support this more general access structure.

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). We will formalize our assumptions regarding TPMs in Section 4.

To support the cryptographic operations required by the TPM specification, TPMs are 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 [39] was one of the first such secret sharing schemes, and is secure in the information theoretic sense (not depending on any complexity assumptions). Other secret sharing variations, using both thresholds and more general notions of access control structures, include those by Feldman [16], Blakley [5] and Krawczyk [30]. 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, similar to a verifiable escrow scheme — 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 verifiable escrow and more involved protocols, such as the fair exchange protocol that we describe in Section 4.4, where a party uses the condition to commit to certain values when starting the verifiable escrow 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 SHARE(r, k, n) for some secret sharing scheme.

Implementation note: For these operations, a TPM could use Shamir's secret sharing scheme [39], and for the `TPM_MakeRandomSecret` operation the TPM would generate k random values $c_0, \ldots, c_{k-1} \in \mathbf{Z}_q$ which would be treated as coefficients of a degree $k-1$ polynomial $p(x) = c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \cdots + c_1x + c_0$, where c_0 is the random secret r, and the shares are just $s_i = p(i)$. The TPM would simply need to keep the k coefficients so that they can be located from the assigned secret handle $secHandle$, and the computations required are similar to existing capabilities within a TPM. We conjecture that real applications of this technique would use small values of k (two or three), and so this is in fact quite efficient to implement within a TPM. Moreover the share generation operation requires only polynomial evaluation, and can be done with $k-1$ modular multiplications and additions by using Horner's rule [13, Section 30.1].

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_MakeRandomSecret1`$(hCond) \rightarrow (secHandle, r)$: 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_EncryptShare1`$(secHandle, AIKHandle, PK) \rightarrow$
$Sign_{AIK}(E_{PK}(hCond \parallel r), PK, g^r, hCond)$: 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 [8]. 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 (of course, the sender can also reveal s to the receiver at any time). 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 [8], 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 4.1 (Generic Verifiable Group Encryption) *Let $\mathbf{R}$ be a relation and define language $L_{\mathbf{R}}$ by $L_{\mathbf{R}} = \{x \mid \exists s : (x, s) \in \mathbf{R}\}$. 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 4.1 describes 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. $\Gamma = \{\tilde{\mathcal{D}} \,|\, \tilde{\mathcal{D}} \subseteq \mathcal{D}$ 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 solution.

Here we note that, like any physical device, TPMs are not perfectly *tamper-proof*, so to keep this clear we refer to *tamper-resistance* of TPMs. However, physical protection of secrets within a TPM is an explicit design goal of TPMs, reflected in the requirement from the TPM Protection Profile [23] that "the [TPM Security Functions] shall resist physical manipulation and physical probing." While other aspects of TPM security have been shown to be vulnerable to physical attacks, such as attacks on system integrity measurements [29], our only physical security assumption is the basic protection of secrets that is described in the Protection Profile. TPMs manufactured today (STMicro, Atmel, Broadcom, Infineon, etc.) have a high level of tamper resistance with regard to protection of secrets, so we believe that our security assumption is realistic.

Definition 4.2 *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 Section 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 4.1. 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 Section 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. We will prove that this scheme is a secure verifiable group encryption scheme in Section 4.2, but first we demonstrate a useful application.

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 several signature schemes, including Digital Signature Standard (DSS) signatures [18], but for concreteness we instantiate it here with Schnorr signatures [38]. Techniques for using other signature schemes are similar to the techniques described in Section 4 of Asokan *et al.* [1]. Note that the TPM operations used are 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. 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, which we formally establish in the next section.

4.2 Security Properties

In this section we prove that VESENDER and VERECVERIFY provide a secure verifiable group encryption scheme.

Theorem 4.1 *Let* $\mathbf{R}$ *be a relation such that* $\mathbf{R} \subseteq (\mathbf{Z}_p, \mathbf{Z}_q)$ *is defined so that* $\mathbf{R} = \{(g^z, z) \,|\, z \in \mathbf{Z}_q\}$. *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: We consider the three properties required by Definition 4.1.

1. *Completeness*: Follows directly from the protocol.

2. *Soundness*: First, assume that all C_i's were produced by a valid TPM using the `TPM_EncryptShare` function (we will return to this assumption later), and thus we are guaranteed that proxy i can use value C_i to recover its correct share s_i of the secret s. Recall that in our setting, the monotone access structure Γ from Definition 4.1 is defined to be a threshold access structure, where any $\tilde{\mathcal{D}}$ from the Soundness condition of the definition is simply a set of k or more proxies. For any such $\tilde{\mathcal{D}}$ we select k corresponding C_i's in the first step of the VERECOVER function, use the proxies to decrypt the shares from which the secret sharing RECOVER function recovers the secret r such that $t = g^r$. By the reasoning following the algorithm statement we will always recover a valid secret s.

   Next, consider our assumption that all the C_i's were produced by a valid TPM. Soundness requires that $\mathcal{B}$ from the protocol pass the VERECVERIFY test, which means (by parts 1 and 4 of the Trusted Platform Security Assumption) that all C_i's have been signed by valid a AIK, i.e., a key that only exists in usable form inside the TPM, and hence is unavailable to an attacker. Therefore, the only way an adversary could produce a C_i that was not produced by a valid TPM is to forge a signature, which can only be done with negligible probability by Part 3 of the Trusted Platform Security Assumption. The soundness property follows.

3. *Computational Zero Knowledge*: For any probabilistic polynomial time (PPT) verifier $\tilde{V}$, we need a PPT simulator $S_{\tilde{V}}$ that can simulate $\tilde{V}$'s view of the transaction with just the common input variables and without access to the prover. The common input variables consist of the variables that are given to both the prover and verifier prior to the interaction, and include the general system parameters as well as the public keys of n proxies, $PK_1, PK_2, \ldots, PK_n$ and the value x that is the one-way image of secret s, as $x = g^s$ (note that s is only known to the real prover, not the verifier or the simulator), as well as any condition $hCond$.

   The simulator does the following: First it generates two random keypairs for signing, one that is generated in exactly the same way that a TPM generates an AIK, with secret signing key SK_A and public verification key VK_A, and a second keypair that is generated exactly the same way as a PrivacyCA's signing key, with signing and verification keys SK_C and VK_C, respectively. The simulator then computes a fake certificate for VK_A by signing the certificate form using SK_C, and we denote the result with notation $Cert_{VK_C}(VK_A)$ — note that the only difference between these keys and the keys used by a real TPM is that the key SK_C does not belong to a legitimate PrivacyCA, so we have

violated Trusted Platform Security Assumption part 4 (we'll argue later that this does not matter when considering the computational zero knowledge property).

Next, the simulator selects $n+1$ random values $d, s_1, s_2, \ldots, s_n \in_R \mathbf{Z}_q$. The simulator computes $t = g^d x^{-1}$ and then creates values $C_i = Sign_{SK_A}(E_{PK_i}(hCond \parallel s_i), PK_i, k, n, t, hCond)$ for $i = 1, 2, \ldots, n$, where PK_i is the public encryption key of proxy i. Finally, the simulator creates the tuple $\mathcal{B} = \langle d, x, t, C_1, C_2, \ldots, C_n, Cert_{VK_C}(VK_A)\rangle$ which it gives to $\tilde{V}$. Since t was computed so that $g^d = xt$, this tuple will be verified by the VERECVERIFY function. Now let $\tilde{D}$ be the subset of proxies mentioned in the definition of the computational zero knowledge property, and so $\tilde{D} \notin \Gamma$, which means that if $j = |\tilde{D}|$ where $j < k$. Using the set $\tilde{D}$ of proxies, we can decrypt j of the ciphertexts contained in the C_i's, say $s_{i_1}, s_{i_2}, \ldots, s_{i_j}$. Consider what $\tilde{V}$ would see from a real prover at this point: j shares of secret r, as well as t which depends on r — however, since we use a secret sharing scheme with threshold k and $j < k$, the decrypted shares are random and independent of both each other and r. This is the exact same as the distribution when we use the simulator. In addition to the decrypted shares $s_{i_1}, s_{i_2}, \ldots, s_{i_j}$, which are random and independent, $\tilde{V}$ also has access to all ciphertexts C_i. When $\tilde{V}$ interacts with the simulator, the corresponding plaintexts are simply random values, so do not contain any information related to the secret r. When $\tilde{V}$ interacts with the real prover, the corresponding plaintexts are shares of r, but since the encryption scheme is CCA secure this is indistinguishable to $\tilde{V}$ from the simulator case. Therefore, the simulator presents a view to $\tilde{V}$ that is indistinguishable from the prover, and hence our protocol is zero-knowledge.

The above arguments show that the desired properties of a secure verifiable group encryption scheme are all met. ∎

4.3 Comparison to Non-Trusted Platform Protocols

We now briefly compare the costs of our protocol to two prior verifiable encryption protocols: that of Camenisch and Damgard [8] (the "CD" protocol), and Camenisch and Shoup [9] (the "CS" protocol). We first consider the CD protocol, as it is the only one to support verifiable group encryption, and hence is the most directly comparable to our TPM-based verifiable group encryption protocol.

We compare costs in Table 1. Our TPM-based protocol is patterned after the CD protocol, with the cut-and-choose proof replaced by a TPM signature, so the costs can be compared fairly simply. Looking at one round of the CD protocol, the cut-and-choose technique requires that the prover actually create two sets of values: one will be used to answer a challenge to honesty from the verifier, and the other to actually use in the encryption. Because of this duplication of effort, the CD protocol requires twice the number of encryptions as our protocol; however, trust in our protocol is built by the hardware signing values, so we must do n signatures, resulting in the total cost being similar between our protocol and one round of the CD protocol. However, for the CD protocol this round must be repeated in order to build trust: a dishonest prover can get away with cheating on a single iteration of the prover's 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 when there is a single proxy, removing the cut-and-choose and reducing the amount of interaction substantially; however, the CS protocol is not directly applicable to the multiple-proxy verifiable group encryption problem. We compare all three schemes in this restricted single-proxy setting in Table 2. While the CS protocol does not require the high interaction of the cut-and-choose proof used by the CD protocol, it is nonetheless still an interactive protocol. The TPM-based

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

Table 1: Comparison of the cost of Camenisch-Damgard (CD protocol) to our TPM-based solution for n proxies, with a threshold of k. Note that the Camenisch-Damgard protocol has a probability of 1/2 of being "tricked" each round, so is repeated 30-100 times in order to build up trust in a statistical sense.

	CD protocol	CS protocol	Our Solution
	Interactive	Interactive (3 rounds)	Non-interactive
	Cost for *One Round* (of 30-100)	Cost for Full Solution	Cost for Full Solution
Prover:	2 encryptions 1 mod powering	1 non-std encryption 10 mod powerings	1 encryption 1 mod powering 1 signature
Verifier:	1 encryption 1 mod powerings	13 mod powerings	2 sig. verifies 1 mod powerings
Recovery:	1 decryption		1 decryption

Table 2: Comparison of major costs of single-proxy solutions: Camenisch-Damgard (CD protocol), Camenisch-Shoup (CS protocol), and our TPM-based solution.

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.

In addition, the CS protocol requires the use of a custom-designed encryption scheme, which is less well-studied than the standard encryption schemes which can be used in the CD protocol or in our solution. While providing some nice properties, this new encryption scheme requires significantly more work than traditional public key encryption schemes: Modular operations are performed modulo n^2, so are approximately twice as large, and a single encryption requires 5 modular powerings and several modular multiplies, compared to a single modular powering for RSA-based methods (both methods also require some hashing and lower-complexity operations as well).

As mentioned in Section 4.1, we would again like to stress the point that the Schnorr signature scheme is not an integral part of our protocol and we have shown it just as an example. In place of Schnorr, the user is free to use any other supported signature scheme, and our protocol will work just as well. Furthermore, since our verifiable encryption scheme uses generic operations that are not tied to the Schnorr signature scheme, the security of the verifiable encryption of the signature is in no way dependant on the security of

the underlying signature scheme.

4.4 Fair Exchange of Digital Signatures

Fair exchange is an important application of the verifiable group encryption protocol outlined above, where additional operations are included in order to synchronize the actions and ensure that the fair exchange property is satisfied. One of the open problems left by Asokan *et al.* [1] was to improve the efficiency of the verifiable escrow operation by avoiding cut-and-choose. Camenisch and Shoup showed that this was indeed possible by introducing a new encryption scheme [9]. By using TPMs, we have shown an alternative method that can make use of standard, well-studied encryption schemes, and further can make the verifiable escrow operation fully non-interactive.

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 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 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 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. For ease of reference, the adapted functions are given in Appendix A.

The fairness of this protocol is established following reasoning similar to that in Asokan *et al.* [1], using Theorem 4.1 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 4.2 *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 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. Under sensible assumptions about the construction of a TPM, we proved that our protocols are secure. One interpretation of this is that we replace the statistical trust gained by repeating multiple rounds of a cut-and-choose protocol with a degree of trust in the manufacturer of the TPM.

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] N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. In *Advances in Cryptology – EUROCRYPT '98*, pages 591–606, 1998.

[2] F. Bao, R. Deng, and W. Mao. Efficient and Practical Fair Exchange Protocols with an Off-line TTP. In *Proceedings of the* 19^{th} *IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[3] B. Barak, M. Prabhakaran, and A. Sahai. Concurrent Non-Malleable Zero Knowledge. In *Proceedings of the* 47^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 345–354, 2006.

[4] M. Bellare and A. Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and concurrent attacks. In *Advances in Cryptology - CRYPTO '02*, pages 162–177. LNCS Springer-Verlag, 2002.

[5] G. Blakley. Safeguarding Cryptographic Keys. In *AFIPS National Computer Conference*, pages 313–317, 1979.

[6] Stefan Brands. Untraceable Off-line Cash in Wallets with Observers. In *CRYPTO '93*, pages 302–318. Springer-Verlag, 1993.

[7] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *ACM Conference on Computer and Communications Security*, pages 132–145, 2004.

[8] J. Camenisch and I. Damgard. Verifiable Encryption, Group Encryption, and their Applications to Separable Group Signatures and Signature Sharing Schemes. In *Advances in Cryptology - ASIACRYPT '00*, pages 331–345. LNCS Springer-Verlag, 2000.

[9] Jan Camenisch and Victor Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Advances in Cryptology - CRYPTO '03*, pages 126–144, 2003.

[10] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[11] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New Constructions for UC Secure Computation using Tamper Proof Hardware. In *EUROCRYPT '08*, pages 545–562, 2008.

[12] David Chaum and Torben Pryds Pedersen. Wallet Databases with Observers (extended abstract). In *Advances in Cryptology - CRYPTO 92*, pages 89–105. Springer-Verlag, 1992.

[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C.Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, Second edition, 2002.

[14] Ronald Cramer and Torben J. Pedersen. Improved Privcy in Wallets with Observers (extended abstract). In *EUROCRYPT '93*, pages 329–343. Springer-Verlag, 1993.

[15] Ivan Damgard, Jesper Buus Nielsen, and Daniel Wichs. Isolated Proofs of Knowledge and Isolated Zero Knowledge. In *EUROCRYPT '08*, pages 509–526, 2008.

[16] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *FOCS '87*, pages 427–437. IEEE Computer Society, 1987.

[17] A. Fiat and A. Shamir. How to prove to yourself: practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86*, pages 186–194. LNCS Springer-Verlag, 1987.

[18] FIPS 186. Digital signature standard. Federal Information Processing Standards Publication 186, U.S. Department of Commerce/NIST, 1994.

[19] M. Franklin and M. Reiter. Verifiable Signature Sharing. In *EUROCRYPT '95*, pages 50–63. LNCS, Springer-Verlag, 1995.

[20] R. Gennaro. Multi-trapdoor Commitments and their Applications to Proofs of Knowledge Secure under Concurrent Man-in-the-Middle Attacks, 2004.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Jounal of Computing*, pages 186–208, 1989.

[22] J. Groth. A verifiable Secret Shuffle of Homomorphic Encryptions. In *Proceedings of the* 6^{th} *International Workshop on Theory and Practice in Public Key Cryptography*, pages 145–160, 2003.

[23] Trusted Computing Group. Protection profile — PC client specific trusted platform module. Available at https://www.trustedcomputinggroup.org/specs/TPM/. TPM Family 1.2; Level 2.

[24] Trusted Computing Group. Trusted Platform Module Specifications – Parts 1–3. Available at https://www.trustedcomputinggroup.org/specs/TPM/.

[25] Vandana Gunupudi and Stephen R. Tate. Random Oracle Instantiation of Distributed Protocols using Trusted Platform Modules. In *21st International Conference on Advanced Information Networking Applications*, pages 463–469, 2007.

[26] Vandana Gunupudi and Stephen R. Tate. Generalized Non Interactive Oblivious Transfer using Count-Limited Objects with Applications to Secure Mobile Agents. In *Financial Cryptography*, pages 98–112, 2008.

[27] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standard smartcards. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 491–500, 2008.

[28] Jonathan Katz. Universally Composable Multi Party Computation using Tamper-proof Hardware. In *EUROCRYPT '07*, pages 115–128, 2007.

[29] Bernhard Kauer. OSLO: Improving the security of trusted computing. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–9, 2007.

[30] Hugo Krawczyk. Secret sharing made short. In *Advances in Cryptology - CRYPTO '93*, pages 136–146. Springer-Verlag, 1993.

[31] N. Low, D. Maxenchuck, and S. Paul. Anonymous Credit Cards. In 2^{nd} *ACM Conference on Computer and Communications Security*, pages 108–117, 1994.

[32] N. Low, D. Maxenchuck, and S. Paul. Anonymous Credit Cards and their Collusion Analysis. In *IEEE/ACM Transactions on Networking*, pages 809–816, 1996.

[33] G. Medvinsky and B.C.Neuman. NetCash: A Design for Practical Electronic Currency on the Internet. In *Proceedings of the* 1^{st} *ACM Conference on Computer and Communications Security*, volume 1993, pages 102–106, 1993.

[34] Tal Moran and Gil Segev. David and Goliath Commitments: UC Computation for Asymmetric Parties using Tamper-proof Hardware. In *EUROCRYPT '08*, 2008.

[35] G. Poupard and J. Stern. Fair Encryption of RSA Keys. In *EUROCRYPT '00*, pages 173–190. LNCS, Springer Verlag, 2000.

[36] I. Ray and I. Ray. Fair Exchange in E-Commerce. In *Proceedings of ACM SIGEcom Exchange*, pages 9–17, 2002.

[37] L.F.G. Sarmenta, M. van Dijk, C.W. O'Donnell, J. Rhodes, and S. Devadas. Virtual Monotonic Counters and Count-Limited Objects using a TPM without a Trusted OS. In *STC '06*, 2006.

[38] C. Schnorr. Efficient Signature Generation by Smart Cards. *Jounal of Cryptology*, pages 161–174, 1991.

[39] A. Shamir. How to share a secret. *Communications of the ACM*, pages 612–623, 1979.

[40] Markus Stadler. Publicly Verifiable Secret Sharing. In *Advances in Cryptology - EUROCRYPT '96*, pages 190–199, 1996.

[41] J. Zhou and D. Gollman. A Fair Non-Repudiation Protocol. In *Proceedings of the* 5^{th} *IEEE Symposium on Security and Privacy*, pages 55–61, 1996.

A ABORT and RESOLVE functions in the Fair Exchange protocol

The following functions are used in the fair exchange protocol of Section 4.4 in order to coordinate actions with the trusted third party when such interaction is necessary. These are straightforward adaptations of the functions from Asokan *et al.* [1], and are provided here for ease of reference.

Algorithm A-ABORT(r, m_B, y_B)
$v \leftarrow f(r)$
if $(\mathsf{no\text{-}abort}, v) \in S$ **then return** "Error"
if $(\mathsf{deposit}, v, m_B, y_B, \sigma_B) \in S$ for some σ_B **then return** σ_B
Add (abort, v) to S
return "Successfully aborted"

Algorithm A-RESOLVE$(r, \alpha, \beta, c_B, m_B, y_B, m_A, y_A)$
$v \leftarrow f(r)$
if $(\mathsf{abort}, v) \in S$ **then return** "Error"
Add $(\mathsf{no\text{-}abort}, v)$ to S
Decrypt α with condition (v, m_B, y_B) to obtain σ_A
if decryption failed **or not** SCHNORRVERIFY(σ_A, m_A, y_A) **then return** $\perp$
$hCond \leftarrow h(\langle v, \alpha, m_A, y_A \rangle)$
return SIGRECOVER$(c_B, \beta, hCond)$

Algorithm B-RESOLVE$(v, \alpha, m_A, y_A, m_B, y_B, \sigma_B)$
if $(\mathsf{abort}, v) \in S$ **then return** "Error"
Add $(\mathsf{no\text{-}abort}, v)$ to S
if not SCHNORRVERIFY(σ_B, m_B, y_B) **then return** $\perp$
Decrypt α with condition (v, m_B, y_B) to obtain σ_A
if decryption failed **or not** SCHNORRVERIFY(σ_A, m_A, y_A) **then return** "α is an invalid escrow"
Add $(\mathsf{deposit}, m_B, y_B, \sigma_B)$ to S
return σ_A