The Dark Side of Threshold Cryptography

Shouhuai Xu[1] and Moti Yung[2]

[1] Laboratory for Information Security Technology
Department of Information and Software Engineering
George Mason University, Fairfax, VA 22030, USA
sxu1@gmu.edu
[2] Certco, New York, NY, USA
moti@cs.columbia.edu

"The whole is more than the sum of its parts"

Abstract: It is typical for a cryptographic technology to be useful in its primary goal and applications, yet to exhibit also a dark side, namely to allow abuses in some other situations. Examples are subliminal channels in strong (randomized) signature schemes, employing authentication for encryption, kleptography exploiting strong randomness, etc. *Threshold cryptography* was introduced to realize better security and availability. However, its "dark side" has never been addressed seriously. We investigate some possible abuses of threshold cryptography which result from users not possessing the entire private key due to threshold splitting. This is a deficiency which can hurt de-commitment in procedures like "contract signing" and nullify non-transferability properties. To attempt solving the problem, one may suggest to assure that the user has full control of his private key via a zero-knowledge confirmation. However, advances in cryptography itself defeat this, since the Completeness Theorem for secure computations implies that servers in possession of shares of a key can answer on behalf of the "virtual holder" of the entire private key, without compromising the secrecy of their shares. We are then forced to look at more physical limitations of the setting. We propose a notion we call Verifiable Secret Non-Sharing (VSNS) where we can replace the strong (i.e., less realistic) physical isolation assumption (namely, a Faraday's cage) with a more realistic *timing* assumption . We then introduce a new class of "combined software engineering and cryptography" adversarial capability, which employs software preprocessing and cryptography in breaking all previous suggestions to our problem. It seems that the adversary is so powerful that we have to rely on certain tamper-resistant device to block it. We show how to prevent a malicious participant from compromising the secrecy of the provers' secret keys in this case.

Keywords: contract signing, non-transferable proofs, undeniable signature, abuse of protocols, threshold cryptography, non-sharing, CA, key certification

1 Introduction

Threshold cryptography (see [DF89,DF91]) distributes a key among a plurality of servers and requires a quorum of servers to sign/decrypt. It has become a

major tool in realizing better security (i.e., more than a single server need to be compromised) and availability (i.e., "single point of failure" devoid). The potential "dark side" of threshold cryptography (e.g., how and when one can abuse it) has not been addressed thoroughly. A source of potential abuses is the fact that third parties do not know whether and how a key is shared due to the transparent nature of the sharing.

The problem is interesting because it is absolutely (i.e., provably) impossible to resolve the sharing issue by a user proving (via traditional protocols and message exchanges) that he knows the key in its entirety. This general obstacle is a result of the Completeness Theorem of secure distributed computation [Y82,GMW87] where users sharing an input can always produce a joint output without revealing the private inputs to each other. In spite of this, we are able to deal with it in the context of designated verifier proofs [JSI96], contract signing [GJM99], and chameleon signatures [KR97] by imposing certain operational restrictions on the model.

1.1 The Problem

The notion of *designated verifier proofs*, due to Jakobsson et al. [JSI96], was motivated to prevent the transferability abuse [DY91, J94] of the verification process in the undeniable signature schemes of Chaum et al. [CvA89]. Specifically, Alice (the prover) proves the statement "either Θ (technically, the corresponding proof of the predicate Θ) is true, or I am Bob" instead of proving Θ directly. Upon seeing such a proof, Bob (the verifier) certainly trusts that Θ is true yet Cindy has no reason at all to believe it. (Retrospectively, the non-transferability of such a proof originates from the *chameleon commitment* schemes introduced in Brassard et al. [BCC88].) Along this way, Garay et al. [GJM99] introduced a new type of digital signature (i.e., *private contract signature*) based on which they proposed an optimistic contract signing protocol with the so-called abuse-freeness property. Abuse-freeness is critical to the fairness of contract signing protocols, yet all of the previously proposed practical solutions are not abuse-free [GJM99]. Also, Krawczyk et al. [KR97] proposed the notion of *chameleon signature* schemes that provides an undeniable commitment of the signers to the contents of the signed documents (as regular digital signatures do) but, at the same time, does not allow the receiver of the signature to disclose the signed contents to any other party without the signer's consent (i.e., non-transferability). Technically, this property comes from the underlying *chameleon hash* [KR97] functions which can be implemented using either the claw-free pairs of trapdoor permutations [GMR88] or the chameleon commitment schemes [BCC88].

The problem we deal with has not escaped the designers. In fact, it has been explicitly [JSI96, GJM99] and implicitly [KR97] mentioned that the assumption "the secret key of the signature receiver is known to himself" is vital to the usefulness of those proposals. For example, if the receiver Bob holds a pair of private and public keys (x, g^x) in the settings of [JSI96, KR97, GJM99] such that x is cooperatively generated by Cindy and himself, the intended non-transferability (e.g., of the chameleon hash) or abuse-freeness is completely broken. This is so since seeing the proof "either Θ is true, or I am Bob" is enough to convince Cindy "Θ is true" since Bob himself (without her help) is unable to present the proof "I am Bob". Independently of the importance of the applications, it is really hard to make sure one did not share his private key with any other party in a general sense (ironically, due to the advances of distributed cryptography itself). Consequently, the colluding provers are able to succeed in convincing

the verifier in *any* cryptographic protocol (e.g., zero knowledge proof, digital signature, decryption, commitment) that the intended secret key is not shared.

So far, the only two potential approaches to dealing with this problem, as far as we know, are due to [JSI96]:

1. When Bob is registering his public key to have it certified, he has to prove knowledge of his secret key to the Certificate Authority, in a setting where he can only communicate with the CA (e.g., a smart-card setting).
2. When registering his public key, Bob presents his secret key to the CA, who then has to be trusted to neither divulge nor prove knowledge of it to anyone else.

It is even reiterated in [GJM99]: The only defense against such an "ultimate" attack, apart from mere social constraints that make very careful planning and administering of attacks "from the beginning of time" less likely to take place, is to make the certification process including a proof of the knowledge for the to-be-certified party's secret key in an "isolated" manner. This naturally invokes the problem of the existence of such an isolated environment (e.g., what is known in the literature as the Faraday's cage [BBD+91]). Unfortunately, the existence of Faraday's cage may be problematic because it is based on a controversial and hard to achieve *physical* (instead of *cryptographic*) assumption. Therefore, a lot of open questions are left to be addressed: (1) Can we replace the physical assumption of the existence of Faraday's cage with a cryptographic counterpart? (2) Are the two potential solutions proposed in [JSI96] really secure? (3) Can we technically prevent such an ultimate attack? As we shall see, we focus on addressing these questions.

1.2 Our Contributions

We present a cryptographic construction to replace the physical assumption of the existence of Faraday's cage [BBD+91]. The function of the newly introduced notion, namely Verifiable Secret Non-Sharing or VSNS for short, is to ensure that the secret key of the certificate holder is not shared with any other party (say, Cindy). The primitive VSNS is based on a new (i.e., the positive use of) *timing* assumption for "secure evaluation of circuits" [Y82, GMW87], which may trigger the research in pursuing their theoretic lower bounds. The timing assumptions combined with the zero-knowledge proof procedure are the crux of this primitive.

We then investigate a harder case and introduce a new class of "combined software engineering and cryptography" attack in which the cheating provers (say, Bob and Cindy) make use of their knowledge to cooperatively *code and debug* a software (or hardware) module which is thus trusted by them (e.g., to accept their secret inputs and random inputs). The module may further be embedded into certain hardware system (e.g., a smart card). This can be done before the start of any protocol. Such a powerful adversary will corrupt the two potential approaches proposed in [JSI96] and even the solution in [BC93] for disrupting the man-in-the-middle attack. In other words, even if we assume the existence of Faraday's cage and that the CA is trusted not to leak any information about the secret keys of the users, the dishonest provers can still succeed in cheating any verifier. In spite of this, we propose a practical solution based on the existence of tamper-resistant hardware that is secure even in the newly introduced adversarial model. The crux of this solution is isolation of trusted components and careful division of trust between parties, based on which we can technically prevent certain participants from compromising the secrecy of the provers' secret keys.

Remark. The main motivation behind this paper is to block the transferability resulted from sharing of certain secrets at the very beginning of system initialization. One may think that transferability is always possible (in small groups) if Bob and Cindy sit together while, for example, Bob receives a proof that "either I am Alice or I can sign m as Bob". However, this is not true. Bob can easily cheat Cindy into accepting a faked proof as follows. Suppose Bob has a brother Bob* whom Bob completely trust, and to whom he gives his private key. Then, Bob asks Cindy to sit near him and see the interaction with the claimed Alice who is in fact impersonated by Bob*. Bob* can always present the proof "either I am Alice or I can sign m as Bob" since he knows Bob's private key. Since Cindy is also smart, she has no reason to be convinced that Alice is at the other end of the communication line. Note that in the application settings we consider here (i.e., designated verifier proof, contract signing, chameleon signatures), Alice does not sign any signature which is non-repudiated in the traditional sense. (In the context of contract signing, Alice only sends her signature after she has been assured that she will also obtain the corresponding signature of Bob.) Moreover, even if Bob's private key is stored in his smart card (i.e., unknown to him), he can ask the application program in the smart card to generate a set of proofs like "either I am Alice or I can sign m as Bob" and then gives them (instead of his private key) to Bob*. Therefore, the transferability by having Bob and Cindy sit together is not a sound solution.

1.3 Related Work

Though our usage of *timing* considerations is different from those in the literature [BD90, BC93, DNS98, K96], there do exist certain correlations. In both [BD90] and [BC93], the authors proposed some solutions (based on *accurate* timing) to blocking man-in-the-middle attacks. For example, the basic idea underlying the "distance bounding" protocol of [BC93] is to let the receiver "rapidly" respond a challenge. As long as each bit of the prover is sent out "immediately" after seeing the challenge bit from the verifier, the delay of the response enables the verifier to compute an upper-bound on the distance. Another use of *timing* assumption (for certain level of synchronization) is to ensure zero-knowledge under concurrent executions [DNS98]. However, they have no consideration of isolation of provers as in our setting. It should also be noted that the *timing* attack introduced in [K96] is a negative use of timing information.

In timing protocols where users may collaborate in the background (e.g., via preprocessing), other considerations must be taken. For example, in the Publicly Verifiable Secret Sharing (PVSS) [S96] replies based on the homomorphism property of the function are used. However, such properties suggest shortcuts for distributed computing. Specifically, the cheating Bob and Cindy can share the random numbers as $k = k_B + k_C$, $w = w_B + w_C$ (refer to [S96] for details). The ElGamal encryption of the secret key $x = x_B + x_C$ can be done in the following way:

$$\begin{aligned} x \cdot y^k &= (x_B + x_C) \cdot y^{k_B + k_C} \\ &= x_B \cdot y^{k_B} + x_C \cdot y^{k_C} + x_B \cdot y^{k_C} + x_C \cdot y^{k_B}. \end{aligned}$$

Thus, we need to avoid such shortcuts which enable "fast collaboration".

[BN00] implemented abuse-freeness adopting an approach (different from [GJM99]) they call "timed commitments". Technically, they allow two parties to fairly exchange RSA or Rabin signatures. Note that our solutions will directly enhance the security of the proposals in [JSI96, KR97, GJM99, GM99].

1.4 Outline

Section 2 is the description of the tools we will use. The new primitive Verifiable Secret Non-Sharing (VSNS) is defined and constructed in section 3. Section 4 introduces a class of "combined software engineering and cryptography" attack. Our tamper-resistant hardware-based solution is presented in section 5. We conclude in section 6.

2 Preliminaries

In this paper we work in the standard setting of discrete log-based cryptosystems. Let q be a large prime, $p = lq+1$ also a prime where $(l, q) = 1$ (e.g., $= 2$), $g \in_R Z_p$ an element of order q, and G_q the group generated by g. We omit the moduli if they are clear from the context. Let $x \in_R Z_q^*$ be Bob's secret key corresponding to his public key $y = g^x \ mod \ p$. Denote $\{0,1\}^*$ the space of finite binary strings and $\{0,1\}^\infty$ the space of infinite ones.

We work in the Random Oracle model [BR93]. For convenience, a random oracle f is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of $f(x)$ uniformly and independently, for every x. Since we never use infinitely long output, this is just for the sake of convenience not to specify the concrete output length. Therefore, we have

$$\begin{aligned} &\Pr[f(a+b) = C] \\ =\ &\Pr[f(a+b) = C | f(a) = A] \\ =\ &\Pr[f(a+b) = C | f(b) = B], \end{aligned}$$

where $+$ could be in natural numbers N rather than in the group (for example) Z_q from which a and b are chosen. When the random oracle is instantiated from cryptographic hash functions as in [BR93], $f(\cdot)$ can be practically computed (i.e., we explicitly distinguish "practical computations" from "efficient computations") provided that the input is known. However, if Bob and Cindy wants to securely (in the sense of [Y82, GMW87, G98]) compute $f(a+b)$ where $a+b$ is unknown to anyone of them, they have to traverse the corresponding circuits gate-by-gate and there are no algebraic shortcuts in the computation (i.e., this is polynomially efficient, yet impractical computation).

3 Verifiable Secret Non-Sharing (VSNS)

Next we develop a primitive combining zero-knowledge proof of knowledge and timing constraints. It is intended to replace the strong physical assumption of the existence of a Faraday's cage.

3.1 The Model

The Participants. Basically and ideally, there are two participants such that the prover Bob intends to apply a "not shared" certificate from the verifier CA. A "not shared" certificate for one's public key g^x is the same as a normal certificate except that it additionally indicates "the private key x of the certificate holder is not shared by the holder with anyone else" (i.e., it is known to the holder). Such a certificate will indicate to Bob's business partners about his credit rank while doing business (like contract signing [GJM99, GM99]) with him. In order to do this, Bob needs to convince the CA that his secret key is known to himself. All the participants are supposed to be of probabilistic polynomial time capability.

The Communication. All the communication channels are public and thus subject to eavesdropping by any party. This is the same as in an interactive zero-knowledge system.

The Adversary. The dishonest Bob manages to obtain a "not shared" certificate for his public key g^x corresponding to his secret key x which is cooperatively generated and thus shared by Cindy and himself. Then, they again cooperate in the proof process with intention to convince (i.e., succeed in cheating) the verifier CA that his secret key is known to him. However, we do assume that Bob will not let Cindy have complete control over his private key x (i.e., self-enforcement).

The Timing Assumption. Let δ and τ be the upper-bound and the lower-bound for the *one-way* communication delay between the prover and the verifier, respectively, γ be the upper-bound for the time complexity of the typical computation of function $f(\cdot)$ when the input is not shared by participants, α be the lower-bound for the time complexity of the secure gate-by-gate evaluation of the circuit(s) when the input is shared by some participants. The timing assumption means that $\alpha > \gamma + \delta - \tau$.

Remark 1. Whether to use the CA off-line (i.e., we trust certificates indicating not-shared private keys corresponding to the certified public keys) or on-line (i.e., we trust third party verification) is a matter of the setting. Since not everyone uses his/her certificate to do business as in [JSI96, KR97, GJM99, GM99], it is completely up to the customer whether or not to apply a "not shared" certificate. Indeed, some users may prefer to share their secret keys with their friends for the sake of better security and availability (e.g., using threshold cryptography). Therefore, this can be viewed as a value-added service of the CA. Technically, either an optional field in the certificate itself (i.e., the syntax approach) or a different certificate signing/verification key (i.e., the semantics approach) can be used to discern whether or not a certificate is a "not shared" one.

2. The reason we need the timing assumption is that Bob has to cooperate with Cindy in computing the answers to the verifier CA's challenges. The intuitive idea is to make positive use of the time complexity in secure multi-party computation [G98]. For example, let $f(\cdot)$ be a publicly specified function instantiation of the random oracle [BR93], $f(x)$ can be done (say) within 1 second for any input x in the domain. If the input x is shared between Bob and Cindy (e.g., $x = x_B + x_C$), in order to securely (in the sense of [G98]) calculate $f(x)$, they have to traverse the corresponding circuit(s) gate-by-gate with an assumed time complexity (say 1 minute). Therefore, Bob and Cindy cannot answer the challenges from the verifier rapidly enough. Moreover, the existence of publicly known time bounds for communication is used to ensure that the time difference in the two computations will not be covered by the time difference between the two communication delays. Consequently, if $\alpha - \gamma$ is large enough, the communication upper and lower bounds need not to be very *accurate* (in contrast with [BD90,BC93] which require more refined time management).

3.2 The Objective

For the sake of simplicity, we consider the two-party case (i.e., the cheating provers Bob and Cindy are trying to convince the honest verifier CA) whereas the extension to multi-party is trivial.

Definition 1. *A VSNS is a protocol for the prover Bob to convince the verifier that he knows the secret key corresponding to his public key (alternatively, he does not share his secret key with Cindy). If Bob is cheating, then Cindy has to be involved in the verification process. A VSNS protocol is secure if it is:*

- **complete***: Any honest prover can always succeed in convincing the verifier by himself without violating the timing constraints.*
- **sound***: No matter what strategy the cheating provers use, if it acts on its own and the verification is intended not to leak the shared secret key, the probability for the honest verifier to accept within the timing constraints is negligible with respect to the security parameter k. In other words, if the verifier CA accepts, then the secret key is known to one of the cheating provers (either before or after the verification process).*
- **proof-of-knowledge***: If the probability of accept is large enough, then with almost the same probability there is an extractor which can get the private key by making use of the prover as a black box.*
- **zero-knowledge***: The protocol is perfectly zero-knowledge (simulatable without access to the secret).*

The definition extends regular zero-knowledge proof of knowledge (taking into account timing constraints) by capturing the idea that if the verifier is convinced then at least one (of Bob and Cindy) knows the secret key in its entirety. Combining with the self-enforcement assumption, we conclude that Bob knows the secret key.

3.3 The Timing Assumption Based VSNS

The protocol is depicted in Figure 1. The intuitive idea is to force the provers to online compute $f(u+v \bmod q)$ and $f(x+u+v \bmod q)$. If the prover is honest, the verifier will not quit the protocol due to time-out.

3.4 Security Analysis

Theorem 1. *The VSNS protocol is secure.*

Proof (sketch) **complete**. This is trivial from the protocol. Remember δ is the upper-bound for one-way communication delay, γ the upper-bound for the time complexity of typical computation of function $f(\cdot)$ when the input is not shared. Therefore, the *timing* assumption will not time-out the protocol if the prover is honest and thus able to present the answer within time $2(\gamma+\delta)$ starting from the point that the verifier sends its message v.

Sound. In order to prove "if the verifier CA accepts, then the secret key is known (either before or after the verification process) to at least one of Bob and Cindy" (combining with the self-enforcement assumption, which also means that Bob knows his secret key), we prove that if Bob and Cindy share his secret key (which means that the verification process preserves this secrecy property), then the verifier CA will reject except with negligible probability.

First, the probability for anyone not knowing the secret x to convince the verifier is at most 2^{-k}. (The standard two-challenge zk-proofs where without knowing the secret, the cheater can prepare to answer only one challenge.)

Second, consider sharing: suppose $x = x_B + x_C$ and $u = u_B + u_C$, where (x_B, u_B) are the shares held by Bob, and (x_C, u_C) are the shares held by Cindy, respectively. There are two cases.

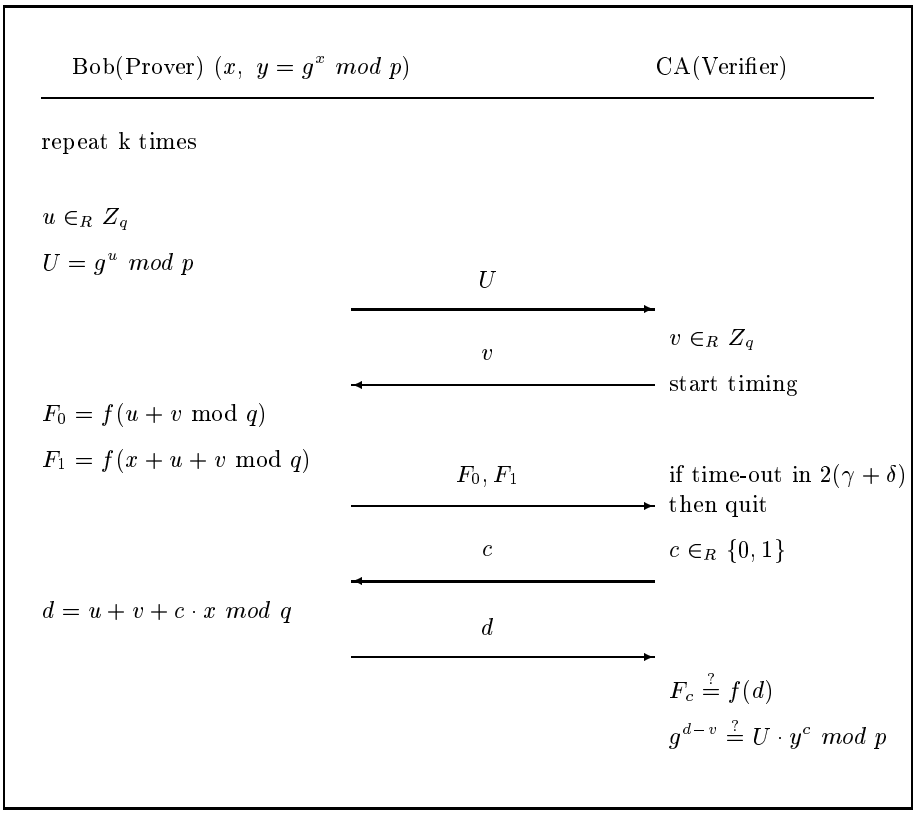


Fig. 1. The VSNS protocol for the proof of the "not shared" knowledge x

- They only compute and present one of F_0 and F_1 (e.g., by computing $u+v = u_B+u_C+v$ and then $f(u+v)$), and choose a random garbage for the other. In this case, they can present F_0 and F_1 before the protocol time-out. However, the probability for them to succeed in cheating the verifier is at most 2^{-k} ($1/2$ per iteration, as mentioned above).
- They compute and present correct F_0 and F_1. There are only three strategies in realizing this. One of these three is the majority strategy (used at least $k/3$ times).
  1. They compute $a = u + v$ and $b = x + a$, then $F_0 = f(a)$ and $F_1 = f(b)$. Obviously, this will leak x even if this stategy is used once, which is exactly what the prover wants to prevent.
  2. They compute one of F_0 and F_1 via distributed gate-by-gate computing, and the other by directly computing (e.g., first $u + v = u_B + u_C + v$ and then $f(u + v)$). In this case, even if the protocol will not time-out, the adversary can succeed in convincing the verifier with probability at most $2^{-(k/3)}$ while preserving the secrecy of the secret key x, if this is the majority strategy.
  3. They compute both F_0 and F_1 via distributed gate-by-gate computing. In this case, the timing assumption $\alpha > \gamma+\delta-\tau$ implies that the cheating provers can not present both F_0 and F_1 before the protocol time-out in any iteration.

Proof-of-knowledge. Using standard arguments, the knowledge extractor asks both questions on an iteration; if the prover has better chance than 2^{-k} then in

a single iteration it needs to be able to answer both challenges $c \in \{0, 1\}$, which gives away the secret key.

Zero-knowledge. We can construct a simulator to obtain a distribution perfectly indistinguishable from the verifier's view in a real runtime. The simulator guesses the verifier's challenge c. If $c = 0$, the simulator chooses $d, v \in_R Z_q$, computes $F_0 = f(d)$ and $U = g^{d-v} \ mod \ p$, chooses at random F_1 from the range of f. If $c = 1$, the simulator chooses $d, v \in_R Z_q$, computes $F_1 = f(d)$ and $U = g^{d-v} \cdot y^{-1} \ mod \ p$, chooses at random F_0 from the range of f. The simulation can be done within expected polynomial time.

□

4 A More Powerful Attack

We now consider a stronger adversary; namely, a new class of "combined software engineering and cryptography" adversarial capability, which is able to break the potential solutions proposed in [JSI96] as well as the timing assumption mentioned above. In the new model, the attackers (say, Bob and Cindy) employ both software (as well as hardware) engineering (read: hacking) and cryptography. Therefore, they can cooperatively *code and debug* some software module (e.g., for cryptographic functions) that is thus trusted by them. (We assume, of course, that the system used by Bob and Cindy to develop the software module is trusted by them. Since Cindy is smart enough not to be fooled, Bob cannot cheat Cindy into using, for example, a malicious compiler as shown in [T84].) The module may be further transplanted into a hardware device (e.g., a smart card). As a result, Bob and Cindy can combine or split keys before or after the process for applying a "not shared" certificate. These attacks suggest the creation of a modified "end-to-end" environment where trust relationships are better handled and better shielded.

4.1 Breaking Faraday's Cage-Based Solution

The module (alternatively, the smart card embedded with the module) is intended to accept the dishonest provers' secret shares and then combine them together before the proving process for applying a "not shared" certificate (as shown in Figure 2). A possible attack scenario could be the following:

1. Bob and Cindy distributively generate Bob's secret key (e.g., $x = x_B + x_C$).
2. They cooperatively generate (i.e., code and debug) a software module which is thereafter trusted by them (e.g., there are no Trojan Horses to leak the intended secret).
3. The trusted module is transplanted into a smart card.
4. The smart card accepts their private shares (i.e., x_B and x_C, respectively) and combines them together to obtain Bob's secret key via $x = x_B + x_C$. Therefore, it is the *trusted* software module or smart card that knows Bob's secret key, but not Bob himself.
5. The smart card applies a "not shared" certificate on Bob's behalf.
6. Finally, the smart card is destroyed (e.g., the software module will automatically destroy the secrets) or kept in a safe such that anyone of them cannot open it.

The attack could happen because the existence of a Faraday's cage does not necessarily mean that the module (alternatively, the smart card equipped with

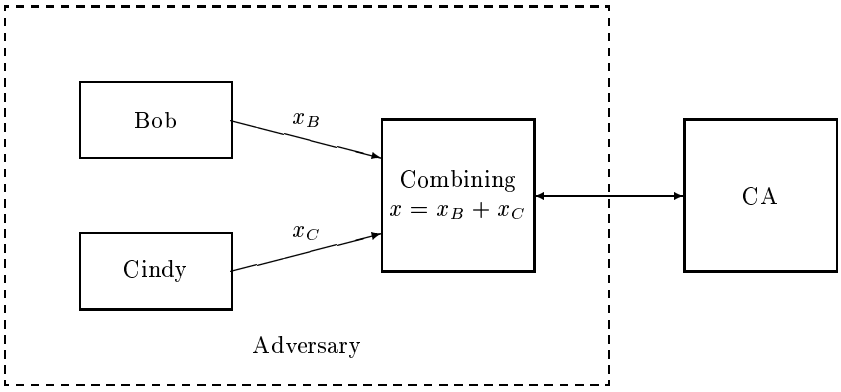


Fig. 2. Breaking Faraday's cage-based solution

the module) can not communicate with Bob and Cindy *before* the commencement of the verification process. Also, the cage (e.g., an isolated device) trusted by the verifier does not necessarily imply that the smart card can automatically be trusted by the verifier.

4.2 Breaking Perfectly Trusted CA-Based Solution

It is not hard to see that, if Bob's secret key is sent to the CA rather than generated by the CA, then the above attack is enough to corrupt the system. Now, we consider a more radical strategy and let the CA generate the users' secret keys. Unfortunately, a system under such a strong trust model is still corruptible (as depicted in Figure 3):

1. Bob and Cindy cooperatively generate (i.e., code and debug) a software module which is thereafter trusted by them (e.g., there are no Trojan Horses to leak the intended secret).
2. The trusted module is transplanted into a smart card.
3. The smart card accepts Bob's secret key and the corresponding certificate from the CA.
4. The smart card shares Bob's secret key via an appropriate secret sharing scheme (e.g., $x = x_B + x_C$). Again, it is the *trusted* software module or smart card that knows Bob's secret key, but not Bob himself.
5. The smart card sends Bob and Cindy their shares (i.e., x_B and x_C respectively) of Bob's secret key and the "not shared" certificate is publicly available.
6. Finally, the smart card is destroyed (e.g., the software module will automatically destroy the secrets) or kept in a safe such that anyone of them cannot open it.

5 A New Solution

In order to deal with the "combined software engineering and cryptography" attack, we present a new solution based on tamper-resistant components (similar

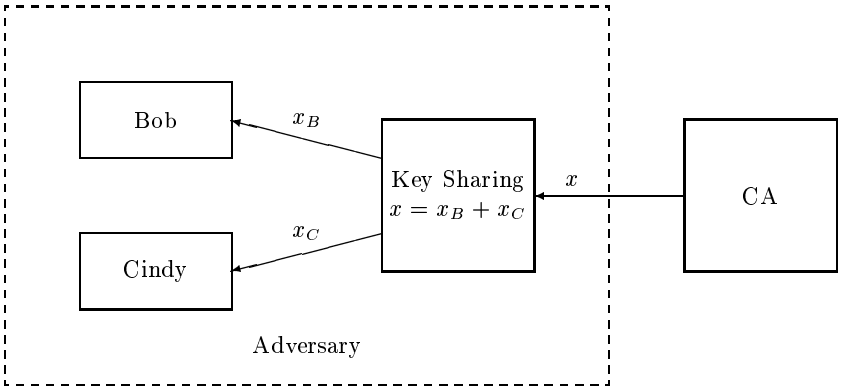


Fig. 3. Breaking trusted CA-based solution

to observers) which are given by the CA to the respective users after certain processing. Due to the introduction of the tamper-resistant components (typically, smart cards), the model has to be refined. As we will see, this (including the analysis of trust relationship implied by the system where various hardware/software components are provided by different parties) is necessary.

Due to the variety in applications, in Section 5.1-5.4, we focus on simple applications like "designated verifier proofs" [JSI96], where a smart card is just used to check whether or not the received proofs are valid. The model and analysis can be seamlessly and modularly incorporated into more advanced applications like [GJM99], where a smart card is also used to generate certain messages (e.g., private contract signatures) which are then forwarded by the card holder to the outside world. In Section 5.5, we present a case study for generating private contract signatures [GJM99].

5.1 The Model

The Participants. Explicitly, there are three categories of participants: the users who apply for "not shared" certificates, the smart cards, and the CA. There is also an implicit participant, the smart card provider, who plays no malicious role in our trust model specified below. The CA gets smart cards from the chosen smart card providers, and gives them to individual users after appropriate processing (e.g., initializing some cryptographic parameters and installing some software programs). As said before, the CA has the incentive to do this because issuing "not shared" certificates is a value-added service. The users then apply for "not shared" certificates by presenting certain "proofs" that are generated by the software program installed in their smart cards.

The Trust. In order to simplify the system, we claim the following trust relationship.

- The smart card hardware is tamper-resistant in the sense that it will erase the secrets stored in it if there is an attempt at breaking into it.
- The smart card software (e.g., operating system) is secure in the following sense: (1) it will not tamper with any application software program installed in it; (2) there is no backdoor for leaking the secrets stored in the smart card; (3) it provides a secure pseudorandom generator, if necessary.

- The CA will not collude with any user to issue him or her a "not shared" certificate without a correct "proof" generated by the corresponding smart card. This also implies that the software program installed in the smart card by the CA (for generating temporary certificates) will not leak any information about the private keys to the card holders.
- The users' computers (including the software programs) are secure. In particular, the randomness output by the software program is guaranteed to be chosen uniformly at random, and there are no Trojan Horses in their computers.

The Communication. Once a user, Bob, has obtained his smart card from the CA, the smart card cannot communicate with the outside world except the corresponding holder, Bob. We assume that the communication channel between Bob and his smart card is physically secure. However, all the other communication channels (including the one between Bob and the CA) are public and thus subject to eavesdropping by any party.

The Adversary. We assume that Bob will not let Cindy have complete control over his private key x (i.e., self-enforcement). Given the above trust model, we consider the following potential attacks.

- The dishonest Bob manages to obtain a "not shared" certificate for his public key g^x corresponding to the private key x that is cooperatively generated by Cindy and himself.
- The dishonest Bob manages to share the private key x corresponding to the public key g^x that is generated by the software program installed by the CA in his smart card and thus certified by the CA via a "not shared" certificate.
- Although the CA is trusted to behave honestly in issuing "not shared" certificates, it may not be trusted in any other sense. For example, the software program installed by the CA for generating "proofs" may not be trustworthy and may intend to leak information via a subliminal channel [S98] or a kleptographic channel [YY97]. As a result, the private key may be completely compromised after further cryptanalysis. For simplicity, we assume that the software program will not adopt the *halting* strategy [D96] to construct a subliminal channel for leaking secret information to the CA. By *halting* strategy we mean that the software program installed by the CA decides whether or not to generate a valid message that is requested by the smart card holder. For example, the software program does generate the requested message only when it has embedded certain information into the subliminal channel known to the CA. We also assume that the software program will not adopt the *delaying* strategy to construct a subliminal channel. By *delaying* strategy we mean that the software program outputs a valid response to a request from the smart card holder only at the time that coincides with a predefined subliminal time channel. For example, a message generated in the morning meaning the bit of 0, and a message generated in the afternoon meaning the bit of 1.

5.2 The Objective

The objective of the protocol is to ensure that one's private key is known to himself (i.e., secret non-sharing), while no information about the private key is leaked to any participant. Since we will adopt a signature scheme for the software program to generate temporary certificates for the users' public keys, we must

guarantee that this process leaks no information about the secrets of the users. For this purpose, we use a subliminal-free signature system, where subliminal-freeness is in the sense that the signer (i.e., the software program installed in a smart card by the CA) cannot abuse the randomness for generating signatures to establish a subliminal channel.

Definition 2. *Suppose the software program installed by the CA in a smart card will output a valid signature within a reasonable delay of computation whenever it is invoked by the card holder. The resulting signature system is subliminal-free, if:*

- *The time at which the software program generates a temporary certificate is completely under the card holder's control.*
- *The to-be-signed messages are guaranteed to be distributed uniformly at random in the corresponding message space.*
- *The randomness used to generate signatures is guaranteed to be chosen uniformly at random. (In the case that the signature generation algorithm is deterministic like the "full-domain hash and RSA"-based signature scheme, this is naturally satisfied.)*

Now we are ready to define the security for a smart card-based solution to ensuring *secret non-sharing*.

Definition 3. *A smart card-based solution to ensuring secret non-sharing is secure, if the followings are satisfied simultaneously.*

- *No dishonest user can obtain a "not shared" certificate for the public key g^x corresponding to a private key x that is cooperatively generated by him and a third party.*
- *No dishonest user can share the private key x corresponding to the public key g^x that is generated by the software program installed by the CA and thus certified by the CA via a "not shared" certificate.*
- *The signature system whereby the software program generates temporary certificates for the users' public keys is subliminal-free.*

5.3 The Solution

In this solution, each participant (e.g., Bob) applying a "not shared" certificate needs to obtain a tamper-resistant smart card from the CA. Each card is equipped with a software program SP provided by the CA, a native temporary pair of public and private keys (pk_{SP}, sk_{SP}), a pair of generators g and h of order q in the subgroup G_q of Z_p^* that is publicly verifiable as in the standard discrete logarithm setting, and a value $g^a h^b$ such that $a, b \in Z_q$. We stress that the discrete logarithm $\log_g h \ mod \ q$ is known to the software program (and possibly also known to the CA) but not any other participant. When Bob applies a "not shared" certificate, he interacts with the software program SP installed in the smart card by the CA to generate a private key x that is unknown to Bob and never departs the card, and to obtain the SP's signature for his public key g^x. Then, Bob presents this signature (i.e., a temporary certificate) to the CA to exchange a "not shared" certificate for his public key g^x. In order to make concrete our solution and to clarify the security of the resulting system, we assume that the software program SP adopts Schnorr signature scheme [S91] to sign temporary certificates. The protocol whereby Bob applies a "not shared" certificate goes as follows:

1. The software program SP sends the equipped $g^a h^b$ as well as its temporary public key $pk_{SP} = g^{sk_{SP}}$ to Bob. If necessary, the software program also sends g, h, p, q to Bob who can thus verify if both g and h are of order q.
2. Bob chooses $z \in_R Z_q$ and sends z to the software program SP. Now Bob can himself compute his public key as $pk'_{Bob}=(g^a h^b)/h^z$.
3. The software program SP calculates the private key for Bob as follows: $x = a + (b - z) \cdot \log_g h \ mod \ q$. Denote $pk_{Bob}=g^x \ mod \ p$. Then, the software program chooses $r \in Z_q$, and sends $R = g^r \ mod \ p$ to Bob. Note that r may not be chosen uniformly at random.
4. Bob chooses $r' \in_R Z_q$ uniformly at random and sends it to the software program SP.
5. The software program SP generates a Schnorr signature for Bob's public key $pk_{Bob} = g^x$ as follows. It computes $r^* = r + r' \ mod \ q$, $c = H(pk_{Bob}, g^{r^*})$, and $d = c \cdot sk_{SP} + r^* \ mod \ q$, where H is an appropriate hash function (i.e., an instantiation of random oracle) with range Z_q. Finally, it sends the signature $SIG_{SP}(pk_{Bob}) \stackrel{def}{=} (pk_{Bob}; c, d)$ to Bob.
6. Bob checks if $pk_{Bob} \stackrel{?}{=} pk'_{Bob}$, $g^d \cdot (pk_{Bob})^{-c} \stackrel{?}{=} R \cdot g^{r'}$, and $c \stackrel{?}{=} H(pk_{Bob}, R \cdot g^{r'})$. If all the verifications pass, Bob sends $SIG_{SP}(pk_{Bob})$ to the CA to exchange a "not shared" certificate $Cert_{CA}(pk_{Bob})$; otherwise, the software program is cheating and the CA is disqualified.

Remark 1. If we adopt an even stronger trust model in which the CA is completely trusted, which means that any software program installed by the CA behaves honestly, then the solution could be simplified since we need not to worry about the existence of any subliminal channel.

2. The software program must commit to various parameters first; otherwise, there could be other subliminal channels. For example, it has two pair of public and private keys and it chooses which one to use based on certain bit of the user's private key.

3. Self-enforcement implies that Bob will not let Cindy have completely control over his smart card. On the other hand, it is unrealistic for Cindy to participate in all the processes (i.e., from obtaining Bob's smart card to verifying a proof designated for Bob) rather than just the process for verifying a designated verifier proof.

5.4 Security Analysis

Proposition 1. *Suppose the software program SP installed by the CA in the smart card will output a valid signature within a reasonable delay whenever it is invoked by the card holder. Then, the resulting signature system is subliminal-free.*

Proof (sketch) By assumption, the smart card will not adopt the *halting* strategy to establish a subliminal channel.

- The time at which the software program generates temporary certificates is completely under the control of the corresponding card holder, since we assume that the software program responds a request from the user within a reasonable delay.
- The to-be-signed messages are guaranteed to be chosen uniformly at random in the corresponding message space. This is because that no matter how the software program chooses g, h, $g^a h^b$, and (pk_{SP}, sk_{SP}), Bob's private key x is uniformly distributed in Z_q since z is uniformly chosen at random, where

$x + z \cdot log_g h = a + b \cdot log_g h$. Therefore, the corresponding public key g^x is also uniformly distributed in G_q.
- The randomness used to generate signatures is guaranteed to be chosen uniformly at random. This is because that no matter how the software program chooses r, $r^* = r + r'$ is uniformly distributed in Z_q since r' is uniformly chosen at random from Z_q. Thus, g^{r^*} is also uniformly distributed in G_q.

□

Theorem 2. *The smart card-based solution to ensuring secret non-sharing is secure.*

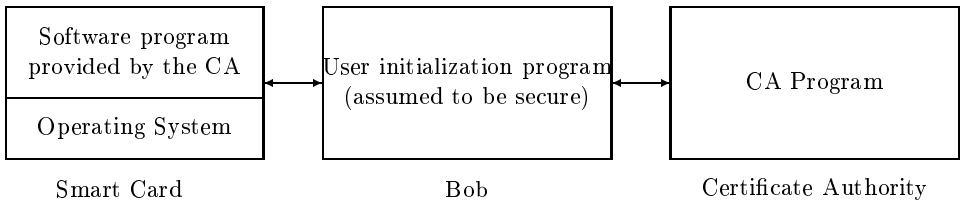
Proof (sketch) We argue that the requirements in Definition 3 are satisfied.

- No dishonest user can obtain a "not shared" certificate for his public key g^x corresponding to the secret x that is cooperatively generated by him and Cindy. This is due to the existential unforgeability of Schnorr signature scheme in the random oracle model, which is used by the software program to generate temporary certificates.
- No dishonest user can share the private key x corresponding to the public key g^x generated by the software program and thus certified by the CA via a "not shared" certificate. This proof is done in two steps.

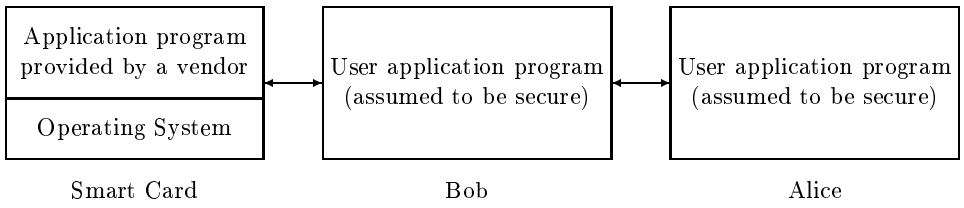
  First, suppose Bob can obtain the corresponding private key x with non-negligible probability, then we can break the discrete logarithm assumption as follows. Suppose we are given $h \in_R G_q$ generated by g and intend to calculate the discrete logarithm $log_g h$. After installing the software program in the smart card, the challenge h is given to the software program that is equipped with a pair of public and private keys (pk_{SP}, sk_{SP}).
  1. The software program chooses $a, b \in_R Z_q$ and sends $g^a h^b$ to Bob.
  2. Bob chooses and sends $z \in Z_q$ to the smart card. Note that z may not be chosen uniformly at random, but this does not matter.
  3. The software program calculates $pk_{Bob} \stackrel{def}{=} g^x = g^a h^b / h^z \ mod\ p$. Then, the software program chooses $r \in_R Z_q$, and sends $R = g^r \ mod\ p$ to Bob.
  4. Bob chooses $r' \in Z_q$ and sends it to the software program.
  5. The software program computes $r^* = r + r' \ mod\ q$, $c = H(pk_{Bob}, g^{r^*})$, and $d = c \cdot sk_{SP} + r^* \ mod\ q$, where H is an appropriate hash function with range Z_q. Finally, the software program sends the signature $SIG_{SP}(g^x) \stackrel{def}{=} (g^x; c, d)$ to Bob.
  6. The signature $(g^x; c, d)$ is always valid, since the software program knows sk_{SP}.

  Therefore, if Bob can obtain the private key x with non-negligible probability, then the discrete logarithm assumption is broken since the software program can calculate $log_g h$ from (a, b) and (x, z).

  Second, we guarantee, when Bob tells Cindy that he has a way to share the private key x corresponding to the public g^x that was certified by the software program as well as by the CA, that Bob really knows the private key x. No matter what algorithm and software they use, finally Bob gets his share x_{Bob} and Cindy gets her share x_{Cindy} with the supposed property (for example) that $x = x_{Bob} + x_{Cindy}$, whereas $g^{x_{Bob}}$ and $g^{x_{Cindy}}$ are known to each other such that $g^x = g^{x_{Bob}} \cdot g^{x_{Cindy}}$. If Bob succeeds in cheating Cindy in the sense that Bob does not know the discrete logarithm of $(g^x)^{-x_{Cindy}}$ with



(a). The interactions in applying a "not shared" certificate



(b). The interactions in generating "private contract signatures"

Fig. 4. The architecture of a "private contract signature" system

respect to base g, it will be on Bob's behalf. Since Cindy is smart, she asks Bob to prove in zero-knowledge about the knowledge x_{Bob} corresponding to $(g^x)^{-x_{Cindy}}$. Therefore, if Bob can share x with Cindy, without each one of them knowing x corresponding to the certified public key g^x, the knowledge extractor will guarantee that x_{Bob} can be known. That is, x can be known.

– The signature system whereby the software program SP generates temporary certificates for the users' public keys is subliminal-free. This is proved in Proposition 1.

□

5.5 Secure Private Contract Signatures: A Case Study for Advanced Applications

In the above we focused on simple applications where smart cards do not send any information to the outside world after initialization. However, for those applications like [GJM99] where a smart card needs to generate *private contract signatures* that will be sent to Bob's business partner (say) Alice over a public channel, there is the possibility that the application program AP provided by an application software vendor may leak information about Bob's private key via a subliminal channel. The architecture for such advanced applications is depicted in Figure 4, where the upper half also corresponds to applications like *designated verifier proofs*.

Although the application program AP for generating *private contract signatures* may not be trusted, we still assume for simplicity that: (1) it will not leak any information about the private keys to the card holders; (2) it will not adopt the *halting* strategy to establish a subliminal channel; (3) it will not adopt the *delaying* strategy to establish a subliminal channel. The rationale of these assumptions can be justified by the fact that the application software provider

might not take the chance to establish a subliminal channel for fearing of losing its revenue.

Private Contract Signatures. Before we define what is a secure private contract signature system, we briefly review what is a private contract signature [GJM99]. Say Alice has public key y_A and private key x_A, where $y_A = g^{x_A}$. Similarly, Bob has a pair of public and private keys (y_B, x_B), and T has a pair of public and private keys (y_T, x_T). In order for Alice to generate a private contract signature on m for Bob, she sends Bob a proof of the statement

"X is a T-encryption of '1' AND I can sign m as Alice
OR
X is a T-encryption of '2' and I can sign m as Bob"

where X is some value, and T-encryption denotes a message encrypted with T's public key. Alice can do this because she can perform a T-encryption of '1' to generate X, and she can sign m herself. After exchanging the private contract signatures in the trustee-invisible scheme, Alice sends to Bob a proof of the statement (we call "proof of decryption" for our purpose)

"Alice: X is a T-encryption of '1' OR T: X is a T-encryption of '1'".

Definition 4. *Suppose the application program in the smart card will output a valid response within a reasonable delay whenever it is invoked by the card holder. A private contract signature generation system is subliminal-free, if:*

- *The time at which the application program generates the private contract signatures is completely under the user's control.*
- *The ciphertext X and the message m are completely under the user's control.*
- *The randomness used to generate private contract signatures and proofs of decryptions is guaranteed to be chosen uniformly at random.*

Now we are ready to define what it means for a smart card-based private contract signature system to ensure secret non-sharing. Note that this requirement does not exist in the context of [GJM99] where the private key is assumed to be known to Bob, while security properties of private contract signature systems proved there should be inherited into the extended private contract signature system we will specify. Since those properties are naturally inherited, we only consider the security requirements inspired by the incorporation of smart cards.

Definition 5. *A smart card-based private contract signature system ensures secret non-sharing, if the followings are satisfied simultaneously.*

- *The process of issuing a "not shared" certificate is secure (see Definition 3).*
- *No dishonest user can share the private key x corresponding to the public key g^x generated by the software program installed by the CA and thus certified by the CA via a "not shared" certificate, even after obtaining polynomially many private contract signatures as well as the corresponding "proofs of decryptions" from the application software in the smart card.*
- *The private contract signature generation system is subliminal-free.*

The Private Contract Signature Generation System. Suppose Bob is to generate a private contract signature on message m for Alice. Bob interacts with the application program AP in his smart card via the following protocol.

1. Bob generates the T-encryption of '2' and then sends the randomness used in generating the encryption to the application program.
2. The commitment for fulfilling "I can sign m as Bob" is computed as follows: the application program chooses and sends $R = g^r$ to Bob; Bob chooses and sends $r' \in_R Z_q$ to the application program; the commitment is defined as $R \cdot g^{r'}$.
3. The application program and Bob collaboratively choose a triple $(c, d_1, d_2) \in_R Z_q^3$. This can be easily done by assuring the application program commits to its randomness first.
4. The application program can generate (via deterministic algorithms) a private contract signature in which (c, d_1) and (c, d_2) are used to simulate the proof that "X is a T-encryption of '1'" and "I can sign m as Alice", respectively. Moreover, the application program can generate (via deterministic algorithms) (c', d_1') and (c', d_2') for the proof that "X is a T-encryption of '2'" and "I can sign m as Bob", respectively.
5. In order to generate the "proofs of the decryptions", Bob and the application program collaboratively generate $(c_2^*, d_2^*) \in_R Zq$ for the proof "T: X is a T-encryption of '2'", then the application program can generate (via deterministic algorithms) (c_1^*, d_1^*) for the proof "Bob: X is a T-encryption of '2'". This can be easily done by assuring the application program commits to its randomness first.

Security Analysis. Similar to Proposition 1, we have

Proposition 2. *Suppose the application program in the smart card will output a valid response within a reasonable delay whenever it is invoked by the card holder. Then, the above private contract signature generation system is subliminal-free.*

Now we are ready to prove that the private contract signature system is secure.

Theorem 3. *The above smart card-based private contract signature generation system ensures secret non-sharing.*

Proof (sketch) We prove that the above system satisfies Definition 5.

- The process of issuing "not shared" certificates is secure. This is exactly the same as in Theorem 2, since it is just a module that is seamlessly integrated into the private contract signature systems.
- Even after polynomially many queries to the application program for generating private contract signatures as well as proofs of decryptions, no dishonest user can share the private key x corresponding to the public key g^x that is generated by the software program installed by the CA and thus certified by the CA via a "not shared" certificate. The proof also has two steps. First, suppose Bob or Cindy can obtain the private key with non-negligible probability, then we can break the discrete logarithm assumption. Basically, we can simulate the real-world process for generating private contract signatures by letting the application program know Alice's private key, and the real-world process for generating proofs of decryptions by letting the application program know T's private key. Second, if Bob can succeed in convincing Cindy via a zero-knowledge proof of knowledge that their shares are correct with respect to the private key x, the knowledge extractor guarantees that x can be calculated. We omit the details.
- The private contract signature generation system is subliminal-free. This is proved in proposition 2.

□

6 Conclusion

We proposed a cryptography-based timing assumption to replace the physical assumption of the existence of a Faraday's cage, based on which we also presented a new notion we call Verifiable Secret Non-Sharing. This solves the problem of secret non-sharing where the parties do not use preprocessing in putting their shares together in order to cheat in the verification. Next, we introduced a class of "combined software engineering and cryptography" attack that is able to corrupt all of the previously proposed solutions. We constructed a tamper-resistant hardware-based solution where the certificate holders are forced not to share their secret keys with any other party. We postulated a concrete model of the system and claimed properties it achieves.

Acknowledgment: We thank the anonymous reviewers for useful comments. Shouhuai Xu was partially supported by an NSF grant to Laboratory for Information Security Technology at George Mason University.

References

[BBD+91] S. Bengio, G. Brassard, Y. Desmedt, C. Goutier, and J. J. Quisquater, Secure Implementation of Identification Systems, Journal of Cryptology, 1991 (4), pp 175-183.

[BC93] S. Brands and D. Chaum, Distance-Bounding Protocols, Eurocrypt'93.

[BCC88] G. Brassard, D. Chaum, and C. Crepeau, Minimum Disclosure Proofs of Knowledge, Journal of Computer and System Science, Vol. 37, No. 2, Oct. 1988, pp. 156-189.

[BD90] T. Beth and Y. Desmedt, Identification Tokens-or: Solving the Chess Grandmaster Problem, Crypto'90.

[BN00] D. Boneh and M. Naor, Timed Commitments, Crypto'00.

[BR93] M. Bellare and P. Rogaway, Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols, ACM CCS'93.

[CvA89] D. Chaum and H. van Antwerpen, Undeniable Signatures, Crypto'89.

[D96] Y. Desmedt, Simmons' Protocol Is Not Free of Subliminal Channels, Computer Security Foundation Workshop'96.

[DF89] Y. Desmedt and Y. Frankel, Threshold Cryptosystems, Crypto'89.

[DF91] Y. Desmedt and Y. Frankel, Shared Generation of Authenticators and Signatures, Crypto'91.

[DNS98] C. Dwork, M. Naor, and A. Sahai, Concurrent Zero-Knowledge, STOC'98.

[DY91] Y. Desmedt and M. Yung, A Weakness of Undeniable Signature, Eurocrypt'91.

[G98] O. Goldreich, Secure Multi-Party Computation, 1998.

[GJM99] J. Garay, M. Jakobsson, and P. MacKenzie, Abuse-Free Optimistic Contract Signing, Crypto'99.

[GM99] J. Garay and P. MacKenzie, Abuse-free Multi-party Contract Signing, DISC '99.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson, How to Play any Mental Game-A Completeness Theorem for Protocol with Honest Majority, STOC'87.

[J94] M. Jakobsson, Blackmailing Using Undeniable Signatures, Eurocrypt'94.

[JSI96] M. Jakobsson, K. Sako, and R. Impagliazzo, Designated Verifier Proofs and Their Applications, Eurocrypt'96.

[K96] P. Kocher, Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS, and Other Systems, Crypto'96.

[KR97] H. Krawczyk and T. Rabin, Chameleon Hashing and Signatures, NDSS'2000.

[S79] A. Shamir, How to Share a Secret, CACM, Vol. 22, No. 11, pp 612-613, 1979.

[S91] C. P. Schnorr, Efficient Signatures Generation by Smart Card, J. of Cryptology, 4(3), 1991, 161-174.

[S96] M. Stadler, Publicly Verifiable Secret Sharing, Eurocrypt'96.

[S98] G. J. Simmon, The History of Subliminal Channels, IEEE Journal on Selected Areas in Communication, vol. 16, no. 4, May 1998.

[T84] K. Thompson, Reflections on Trusting Trust, CACM, Vol. 27, No. 8, pp 761-763.

[Y82] A. Yao, Protocols for Secure Computations (extended abstract), FOCS'82.

[YY97] A. Young and M. Yung, Kleptography: using Cryptography Against Cryptography, Crypto'97.