On the Insecurity of Proactive RSA in the URSA Mobile Ad Hoc Network Access Control Protocol

Stanisław Jarecki and Nitesh Saxena

Abstract— **Access control is the fundamental security service in ad hoc groups. It is needed not only to prevent unauthorized entities from joining the group, but also to bootstrap other security services. Luo, et al. proposed a set of protocols for providing ubiquitous and robust access control (called URSA[1]) in mobile ad hoc networks without relying on a centralized authority. The URSA protocol relies on the new proactive RSA signature scheme, which allows members in an ad hoc group to make access control decisions in a distributed manner. The proposed proactive RSA signature scheme is assumed secure as long as no more than an allowed threshold of participating members is simultaneously corrupted at any point in the lifetime of the scheme.**

In this paper, we show an attack on this proposed proactive RSA scheme, in which an admissible threshold of malicious group members can completely recover the group RSA secret key in the course of the lifetime of this scheme. Our attack stems from the fact that the threshold signature protocol which is a part of this proactive RSA scheme leaks some seemingly innocuous information about the secret signature key. We show how the corrupted members can influence the execution of the scheme in such a way so that the slowly leaked information is used to reconstruct the entire shared secret.

I. Introduction

Access Control in Ad Hoc Groups. Ad hoc groups, such as peer-to-peer (P2P) systems and mobile ad hoc networks (MANETs), are very popular in today's computing, especially in the research community. They lack infrastructure and do not need any trusted authority. Moreover, they are inherently scalable and fault tolerant. Such characteristics of ad hoc groups find many interesting applications in military and commercial settings as well as in emergency and rescue operations. However, their open nature and lack of centralized control result in some security challenges.

The security research community recognized the need for specialized security services in ad hoc groups. *Access Control* is particularly important since most other traditional security services are based upon it. In this context, access control is needed to prevent unauthorized nodes from becoming a part of the group and to establish trust among nodes in the absence of a trusted authority. Access control is also essential to bootstrap other security services, such as secure group communication (group key agreement and key management), e.g., [38], [1] and secure routing, e.g., Ariadne and SRP [21], [32].

A preliminary version of this paper appeared in [23]

Nitesh Saxena (corresponding author) is with Computer Science and Engineering Department, Polytechnic Institute of New York University. E-mail: `nsaxena@poly.edu`.

Stanisław Jarecki is with Computer Science Deaprtment, University of California, Irvine, E-mail: `stasio@ics.uci.edu`

[1]URSA scheme appeared in [28], [25], [24], [29], and the journal [27].

Threshold Cryptography, Threshold and Proactive Signature Schemes. The idea of distributing a cryptosystem so as to secure it against corruption of some threshold, e.g. a minority, of participating players is known as *threshold cryptography* [12], and it is built on the *polynomial secret-sharing* technique of Shamir [37].

A (t, n) threshold signature scheme [13] enables any subgroup of $t + 1$ members in a group consisting of $n > t$ members, to collaboratively sign a message on behalf of that group. This is achieved by secret-sharing the signature key, e.g. the RSA secret key, among the group members, and allowing them to compute a signature on some message via a distributed protocol in which the members use the shares of the signature key instead of the key itself. The scheme is said to be *t-secure* if any coalition of at most t corrupt members is unable to forge a valid threshold signature on any message which honest members would not sign, and *t-robust* if honest group members can efficiently produce a valid signature even in the presence of at most t malicious members. To achieve t-security, a threshold signature scheme must in particular protect the secrecy of the signature key as long as no more than t of the group members are corrupt.

A *proactive* signature scheme [19], based on techniques of proactive secret sharing [20], is a threshold signature scheme which remains secure and robust even if in every *time period*, called "share update interval", a possibly different set of t group members is corrupted. This is achieved by the members periodically updating their shares of the secret signature key via a distributed share update protocol. Such an update protocol should destroy the correlation between secret shares learned by corrupted members in different time periods, so that the scheme can tolerate any number of corruptions throughout its lifetime as long as in any single time period the number of simultaneously corrupted members does not exceed t.

Application of Proactive Signatures to Access Control in Ad Hoc Networks. As suggested by Zhou and Haas [39], proactive signature schemes can be used to implement group access control decisions without relying on a trusted and always accessible group "manager", who makes all admission and revocation decisions on behalf of the group. In many mobile group settings, such manager may be often inaccessible to some subgroup of members. Moreover, in many applications placing all trust in a single entity creates a security liability.

In contrast, the idea of the proactive signature based access control mechanism for ad hoc groups is that any large enough set of members can admit a new group member by using the threshold signature protocol to compute the new member's certificate, and by using a slight variation of the proactive update protocol to give this member his share of the signature key. Similarly, once any large enough set of members agrees to revoke some existing member, the members collectively sign the revocation statement, and trigger the proactive share update protocol to leave the just revoked member without the current share of the signature secret. Luo, et al. [28], [25], [24], [29], [27] first proposed such access control protocol for MANETs based on a new proactive RSA signature scheme (which we analyze in this paper). Saxena et al. [35] implemented similar access control protocol using the proactive DSS signature scheme of Gennaro et al. [17]. The same authors [36] examined the performance of a more efficient access control protocol based on the proactive BLS signature scheme [5] of Boldyreva [3], which relies on elliptic curve cryptography.

However, the common operation of signature *verification* in DSS, BLS, and in all other discrete-log based signature schemes, is orders of magnitude computationally more intensive than the verification of RSA signatures. Therefore, access control protocols based on an efficient provably secure proactive RSA signature scheme offer an attractive alternative to ones based on discrete-log based proactive signatures.

Problems with Currently Known Provably Secure RSA Proactive Signature Schemes. Unfortunately, the most efficient currently known provably secure proactive RSA signature schemes, two schemes by Frankel et al. [15], [14] and a scheme by Rabin [34], are not easily applicable to securing access control in ad hoc groups by the methods described above. The fundamental reason is that the arithmetic operations involved in the RSA signatures seem more difficult to securely distribute than the DSS or the BLS cryptosystems. The difficulty in distributing RSA signatures is caused by the seeming necessity to perform computations on the secret shares modulo $\phi(N) = (p-1)(q-1)$, where $N = pq$ is the RSA modulus. Performing operations modulo $\phi(N)$ is difficult because this number must stay secret for the distributed RSA scheme to be secure. Note that revealing $\phi(N)$ enables anyone to immediately compute the RSA private key $d = e^{-1} \pmod{\phi(N)}$ from the RSA public key (e, N). In contrast, both the DSS signatures and BLS signatures of [5] are based on variants of the discrete logarithm problem where all the moduli used in computations involving secret shares can be made public. In particular, the proactive RSA scheme of [15] is practical only for small groups, while in the other two provably secure proactive RSA schemes known today [14], [34], the members participating in the threshold signature protocol need to reconstruct the secret shares of the group members that are currently inaccessible to them. In this way both protocols essentially equate a temporarily inaccessible group member with a corrupt one, whose secrets might just as well be reconstructed. This is an undesirable feature for asynchronous ad hoc groups where members are often inaccessible to one another. In such settings we need to enable isolated but large enough subgroups of members to operate without reconstructing everyone else's secrets. COCA (a distributed on-line certification authority) [40] employs a modified version of Rabin's RSA scheme which overcomes this problem of availability. However, this scheme, which is based on combinatorial secret sharing as opposed to the additive sharing of Rabin, is applicable only for small groups, because in large groups the number of combinations $\binom{n}{t}$ becomes intractable.

Proactive RSA Scheme Proposed for the URSA Ad Hoc Network Access Control Protocol. In an effort to mitigate the above problem of the known proactive RSA signatures, Luo, et al. [28], [25], [24], [29], [27] proposed a new proactive RSA scheme, geared to wards providing a security service, called "URSA", in MANETs. The original description of this proactive RSA scheme and the URSA application can be found in [28]. Subsequently both the proactive RSA scheme and URSA were described in [25], [24], [29], and most recently in a journal version [27]. The URSA proactive RSA scheme can be applicable to MANETs because it avoids the need to access all shares during the threshold signature protocol. This is because it relies solely on Shamir's *polynomial* secret sharing scheme [37], as opposed to resorting to an additional layer of *additive* secret sharing, as is done by the two most efficient provably secure proactive RSA schemes [14], [34] discussed above. The core of the URSA proactive RSA scheme is the so-called *t-bounded offsetting algorithm* which is used to reconstruct the RSA signature $m^d \pmod N$ from $t+1$ *partial signatures* produced individually by the $t+1$ members participating in the signing protocol.

The first problem with this scheme was pointed out in [31]. Namely, contrary to what the authors of the proposal claimed, their scheme does not provide robustness in signature generation in the presence of t malicious members. Simply speaking, the robustness mechanisms proposed by the authors are faulty because they require certain verification equations to hold even though they in fact do not hold, because the equations involve computation in two different groups (see [31] for more details). Hence, the set of t malicious members can prevent the honest members from efficiently creating a valid signature. However, this robustness problem in the *t-bounded* proactive RSA scheme can be solved, if the secret sharing is performed over a large prime number, instead over the RSA modulus N of the original proposal, coupled with special-purpose zero-knowledge proof protocols for proving equality of discrete logarithms in two different groups [9] and range of a discrete logarithm [6]. Such proof protocols are not very fast, but their expense can be tolerated because they would need to be executed only in the (rare) case of a corrupted member providing an incorrect partial signature to other members. (This will be a rare occurrence because a malicious member behaving in such a manner would be detected by the honest players, and therefore would be subsequently revoked from the group.) We summarize the robustness fix to the URSA proactive RSA scheme in Appendix A. This fix led to an efficient provably secure proactive RSA scheme [22]. We also believe this to be of independent interest.

Since the t-robustness of this scheme can be ensured by the above modifications, there remains a question if the (modified) URSA proactive RSA scheme is *secure* against a coalition of corrupt t members[2] whose goal is not to prevent members from issuing signatures but to learn the secret-shared RSA signature key and to be thus able to forge signatures on the group's behalf. The question is interesting because the proposed URSA proactive RSA scheme, amended as described above, would provide efficiency and functionality advantages over the best known provably secure proactive RSA schemes [15], [14], [34]. However, the answer turns out to be negative.[3]

Our Contribution: An Attack on the URSA Proactive RSA Scheme. We demonstrate the insecurity of the URSA proactive RSA signature scheme by constructing an explicit attack in which the admissible group of t corrupted members colludes in the proactive protocol in such a way so that they reconstruct the whole RSA secret key d after a realistic number of runs of the proactive update protocol and the threshold signature protocol.

Our attack exploits the fact that the *t-bounded offsetting* threshold RSA signature protocol, which is employed in the URSA proactive RSA scheme, leaks certain seemingly innocuous information about the secret signature key. The information that the adversary learns about the secret key in a run of the signature protocol depends on the current sharing of the secret key and on which group of members participates in the protocol. While it is not clear how dangerous this released information is for a single secret sharing, in a *proactive* signature scheme the secret sharing is refreshed with every proactive update, and therefore the released information about the secret key can be different in each update interval. It turns out that the corrupted members can influence an execution of the *update* protocol in such a way that the executions of the *signature* protocol during the subsequent update interval will release information which is both new and correlated with the information the adversary has gained so far. Thus our attack can be seen as a simple search algorithm, where the information learned in a signature protocol tells the adversary which branch to pick next, and the proactive update protocol allows the adversary to pick that branch.

The attack poses a realistic threat. For example, for the threshold size $t = 7$ and for the RSA public key of $e = 65537$ (utilized in the implementation of URSA [25]) and 1024-bit RSA modulus N, our attack needs 163 executions of the proactive update protocol and 1148 runs of the signature protocol to succeed. The attack succeeds assuming that throughout these 163 update periods, the t corrupted players belong to the so-called "update group" of players which play an active role in the proactive update protocol.

However, the URSA proactive RSA protocol is more vulnerable than what is immediately implied by the above attack. First, our attack does not make use of all the information leaked in the signature protocol, thus it is quite possible that another attack, which does utilize all the available information succeeds in recovering the private RSA key even faster and/or succeeds in recovering the key even if, say, only a smaller subset of the special "update group" of players is corrupted. Moreover, even if the attack we describe is slowed down, for example by slowing down the rate of the proactive updates, it recovers $512 + (r - log_{t+1}(e)) * log_2(t + 1)$ most significant bits of d after $r > log_{t+1}(e)$ update rounds, which gives $512 + 3(r - 5)$ MSB bits of d for the above $e = 65537$ and $t = 7$. Therefore our attack raises doubts about the security of the URSA proactive RSA scheme even for smaller number of rounds. It is hard to say much about the security of this protocol, for example after $r = 34$ rounds, because while it is not currently known how to recover the whole RSA key knowing 600 of the MSBs of d, we also do not know any arguments that RSA remains secure with this side information about d revealed, and it would be rather surprising if such arguments existed.

Paper Organization. The rest of this paper is organized as follows: Section II summarizes the notation and introduces some definitions. Section III describes the URSA proactive RSA signature scheme. We then present an attack on this scheme in Section IV. Finally, we elaborate on the implications of our attack for the security of the URSA scheme in Section V.

II. Notation and Setting

An Adversary: An adversary against a proactive signature scheme, and thus also against a MANET group access control mechanism like URSA which utilizes a proactive signature scheme, is able to compromise any set of at most t members in the group in every *update round* (see below). After compromising a member, the adversary learns its corresponding secret share and can force this member to behave arbitrarily in the protocol. We assume the worst case where all the t corrupted members collude, and in fact all corruptions are simply scheduled and controlled by a single entity, called an "adversary" and denoted by $\mathcal{A}$.

Share Update "Round", or "Interval": This is the time period between any two consecutive share update procedures. In the URSA design, the share update protocols occur periodically, for example twice a day. The reason for period execution of this procedure is to re-randomize the secret sharing in such a way that all the currently non-revoked players receive new random shares of the same RSA secret key d. (We describe this protocol in detail in Section III-C.)[4] We will assume that in every share update round, there is a significant amount of signature requests. In fact, our attack requires just t such

[2]In the standard $(t+1, n)$ threshold cryptographic model, any set of $t+1$ out of a total of n members share the ability to perform a cryptographic operation (e.g., signing) in the presence of at most t corruptions. All the earlier versions of the URSA papers [28], [25], [24], [29] work in this standard model. However, the latest journal version of URSA [27] based its scheme on a slightly modified and a non-standard model, wherein any $t+1$ members share the signing operation, but only a maximum of $t-1$ corruptions are allowed. The attack we describe in this paper is based on the standard model. However, as we discuss later in Section V, the URSA scheme appears insecure even in the non-standard model considered in [27].

[3]In particular, although Luo, et al. claim that their scheme is provably secure, the security proofs that appear in [28] are incorrect.

[4]We note that if the update protocol is triggered only periodically as in the URSA design, this has a consequence that only up to t members can be meaningfully revoked from the group within an update interval, because a player is fully revoked only when his share of the secret key d becomes invalid, and that happens only when all the remaining non-revoked players perform a share update protocol. Alternatively, as suggested by [36], an update protocol can also be triggered *reactively*, as an immediate response in the case of revocation of an unusually large number of players.

signature requests per update round to recover the key at a maximum speed.

Table I summarizes the notation used in the rest of this paper.

TABLE I
NOTATION

M_i	group member with unique index i
ss_i	secret share of M_i
t	"adversarial threshold", i.e. the number of tolerated corruptions
n	total number of members M_i
(N, e)	RSA public key
(d, p, q)	RSA private key
TD	trusted dealer
$\mathcal{A}$	the adversary
$\|x\|$	number of bits in a binary representation of x
$lg(x)$	$log_2(x)$
$a = b \pmod{c}$	$a \equiv b \pmod{c}$

III. The Proactive RSA Signature Scheme in URSA

In this section we describe the proactive RSA signature scheme of [28], [25], [24], [29], [27] used in the URSA ad hoc network access control protocol. We will refer to this scheme as an "URSA proactive RSA signature scheme".

A. The Setup Procedure

A trusted dealer TD is involved in a one-time setup to bootstrap the system. The dealer is not required hereafter and in fact is assumed to vanish from the scene, or, equivalently, to erase his memory. TD generates the standard RSA private/public key pair, i.e. it picks two random primes p and q, sets $N = pq$, sets (e, N) as a public key where $gcd(e, N) = 1$, and as a private key it sets a number $d < N$ such that $ed = 1 \bmod \phi(N)$, where $\phi(N) = (p-1)(q-1)$.

Once the standard RSA key pair is chosen, TD secret-shares the RSA secret key d using a slight modification of Shamir secret sharing [37]. Namely, TD selects a random polynomial $f(z)$ over $\mathbb{Z}_N$ of degree t, such that the group secret is $f(0) = d \pmod{N}$. Next, TD gives to each member M_i, for $i = 1, \cdots, n$, a secret share $ss_i = f(i) \pmod{N}$. Notice that the secret d is shared over a composite modulus N as opposed to a prime modulus as in the original scheme of Shamir, but our attack does not depend on what modulus is used in the secret sharing.

B. The Threshold Signature Protocol

The goal of the threshold RSA signature protocol is to generate in a distributed manner an RSA signature $s = m^d \pmod{N}$ under the secret-shared key d. The URSA threshold RSA signature protocol consists of two phases: First each participating member creates its *partial signature* on the intended message and sends it to the signature recipient, and then the recipient locally reconstructs the RSA signature from these partial signatures.

Partial Signature Generation: Let G denote the set of identifiers of the $t+1$ members in the group who participate in the threshold signature protocol. Using polynomial interpolation we can write the secret key d as

$$d = \sum_{j \in G} ss_j \; l_j^{(G)} \pmod{N}$$

where $l_j^{(G)} = \prod_{i \in G, i \neq j} \frac{(-i)}{j-i} \pmod{N}$ Notice that $N = pq$ has only two very large factors, and therefore all the elements $(j-i)$ for $i, j \in G$ will have inverses modulo N. Each member M_j, for $j \in G$, outputs his partial signature $s_j^{(G)}$ on m as

$$s_j^{(G)} = m^{d_j^{(G)}} \pmod{N}, \text{ where } d_j^{(G)} = ss_j \; l_j^{(G)} \pmod{N} \quad (1)$$

Signature Reconstruction: On receiving $t+1$ partial signatures $s_j^{(G)}$ from the $t+1$ group members M_j in G, the signature recipient reconstructs the RSA signature s using the *"t-bounded-offsetting"* algorithm which works as follows. Since $\sum_{j \in G} d_j^{(G)} = d \pmod{N}$ and $0 \leq d_j^{(G)} \leq N-1$ for all j's, therefore

$$d = \sum_{j \in G} d_j^{(G)} - \alpha^{(G)} N \text{ (over the integers)} \quad (2)$$

for some integer $\alpha^{(G)} \in [0, t]$. Equation (2) implies that

$$s = m^d = (\prod_{j \in G} s_j^{(G)}) m^{-\alpha^{(G)} N} \pmod{N}$$

for some integer $\alpha^{(G)} \in [0, t]$. Since there can be at most $t+1$ possible values of $\alpha^{(G)}$, the signature recipient can recover $s = m^d \pmod{N}$ by trying each of the $t+1$ possible values $Y_\alpha = Y(m^{-N})^\alpha \pmod{N}$ for $Y = \prod_{j \in G} s_j^{(G)}$ and $\alpha = 0, \cdots, t$, and returning $s = Y_\alpha$ if $(Y_\alpha)^e = m \pmod{N}$. The most significant cost factor in this procedure is an exponentiation $m^{-N} \pmod{N}$, and therefore the computational cost of the URSA threshold RSA signature protocol for each of the signers and for the recipient is about one full (1024 bit) exponentiation modulo N.

Remark: It is important to note that the above *t-bounded offsetting* threshold RSA signature algorithm **reveals the value of** $\alpha^{(G)}$, which, as will be described in section IV, leaks some information about the secret-shared private key d to an adversary who corrupts t of the players participating in group G. This information leakage in fact exposes the whole proactive RSA scheme to an efficient key-recovery attack.

C. The Proactive Share Update Protocol

The goal of the proactive share update protocol is to re-randomize the secret sharing of the private RSA key d held by the group members. This protocol was first proposed for both proactive secret sharing and for proactive cryptosystems and signature schemes by Herzberg, et al. [20], [19]. The URSA proactive share update protocol is a variant of this protocol which is more efficient, especially in settings where some players can be inactive or temporarily disconnected from others, as in MANETs.

The "Classic" Share Update Protocol. The proactive share update protocol of [20], [19] proceeds as follows (when sharing is done modulo N): Every member M_j chooses a random *partial update polynomial* $\delta_j(z)$ over $\mathbb{Z}_N$ of degree t with the constant term being zero. The sum of these partial update polynomials defines the *update polynomial* $\delta(z) = \sum_{j=1,n} \delta_j(z) \pmod N$. Note that $\delta(0) = 0$. For each pair of members (M_j, M_i), player M_j gives a share $\delta_j(i)$ of his partial update polynomial to M_i. Each member M_i then computes his new secret share (to be used in the threshold signature protocol in the subsequent update interval) as

$$ss_i = ss'_i + \sum_{j=1}^{n} \delta_j(i) = ss'_i + \delta(i) \pmod N$$

where ss'_i is M_i's existing share, i.e. a share this player used in the previous interval. All the information pertaining to this protocol except of the new share ss_i is then erased. Note that if $ss' = f'(i) \pmod N$ for all i then the new secret-sharing polynomial $f(z)$ is defined as $f(z) = f'(z) + \delta(z) \pmod N$, and it is therefore a t-degree polynomial s.t. $f(0) = f'(0) = d \pmod N$.

However, it is important to notice that the new secret-sharing polynomial $f(z)$ is not necessarily a *random* t-degree polynomial s.t. $f(0) = d \pmod N$. This is because a corrupt player M_j can distribute its update polynomial $\delta_j(z)$ only after all the other corrupt players M_i see their shares of all the other update polynomials. In this way, the corrupt players can control the new secret-sharing polynomial $f(z)$ to some degree, by controlling the shares of $f(z)$ held by the corrupted players. In provably-secure proactive schemes that employ this proactive share update protocol, like the proactive DSS or BLS signatures [18], [3], this adversarial ability does not pose any harm. However, as we will see in section IV, this control ability means trouble for the URSA proactive RSA scheme since the information about shared secret d leaked in the threshold signature protocol depends precisely on the shares held by the corrupted players.

The URSA Two-Stage Modification of this Protocol. The URSA proactive RSA scheme utilizes a simple modification of the above share update protocol which improves the protocol's efficiency. This modification can indeed be used to speed up all proactive cryptosystems that use the [20] protocol, as the above mentioned schemes of [18], [3]. The URSA proactive share update protocol consists of two stages. The protocol relies on an existence of a designated group of players Ω, which we will call an "update group", consisting of t group members.[5] The first stage of the protocol proceeds exactly like the above protocol of Herzberg et al., except that only the players Ω participate in it. In other words, players $M_j \in \Omega$ create their update polynomials $\delta_j(z)$, send their shares $\delta_j(i)$ to other members $M_i \in \Omega$, and thus the players in Ω can be said to hold in secret-shared form the update polynomial $\delta(z)$ defined as $\delta(z) = \sum_{j\in\Omega} \delta_j(z) \pmod N$. In the second stage of the URSA proactive share update protocol, the members of the update group Ω provide shares of this update polynomial $\delta(i)$ to *all* remaining (and non-revoked) group members $M_i \notin \Omega$. In this way all group members M_i will get their update share and can compute the new share $f(i) = f'(i) + \delta(i) \pmod N$ as before.

The URSA papers describe two protocols for how these $\delta(i)$ update shares are transferred from the Ω players to their final destinations. Even though the details of the second version of this protocol are a little unclear, these details do not affect the attack we describe in this paper. For completeness, we sketch the two variants as follows: In the first version, each M_i gets its $\delta(i)$ share by communicating with each of the players $M_i \in \Omega$ directly. The players in Ω jointly reconstruct the $\delta(i)$ value by first sharing masking random values among themselves. In the second version, either the $\delta(i)$ update shares or the whole $\delta(z)$ polynomial (this version is not very clear to us), appear to be distributed to the rest of the group members encrypted under the group public key (e, N). Presumably, each member M_i reconstructs his share of this update polynomial $\delta(i)$ by contacting her $t + 1$ neighbors who either decrypt her encrypted share or decrypt the whole polynomial $\delta(z)$ and evaluate it at point i at the same time. It is not clear how this second version can be implemented securely, but the first version is standard and we see no vulnerabilities in it.

Choosing the Ω Update Group. We need to note here that the attack on the URSA proactive RSA scheme that we present in this paper depends crucially on all of the t players in the above update group Ω to be the corrupted players. It is therefore important for the practical feasibility of the attack how this Ω group is decided. From the initial reply of the URSA authors to the attack presented in this paper [26], it appears that the details of how the Ω group is decided are not set in stone in the design of the URSA scheme. This is not surprising since the idea of modifying the Herzberg et al. protocol by delegating the update work to a smaller set of players Ω was introduced for the reasons of efficiency, not security.

The fixes proposed in [26], e.g. choosing the Ω group as the t players with smallest IDs, seem problematic: First, such players would need to be identified in some distributed protocol, and second, the resulting protocol would now be still under the attack, but only if the t players with lowest IDs are corrupted. Moreover, the resulting protocol would then need all these Ω players to be always present and connected, which goes against the URSA philosophy of providing group access control in environments where some players can become inactive and disconnected. Possibly, the tweak that would slow the attack we describe in this paper the most would choose the Ω group differently in every round. This tweak, however, has similar problems: (1) It is not clear how to make sure that the Ω membership rotates without the global knowledge of the current membership list, which would reduce the applicability of the resulting protocol, and (2) the resulting protocol would again need the scheduled players to be up and connected at the right times. In our understanding, the attractive idea of the original philosophy of URSA was that the Ω group can be formed by *any* t players, and moreover that several such groups can be independently created in disconnected fragments of the network.

[5]In some URSA descriptions it seems that the Ω group needs to have $t+1$ and not t members. We believe that t members is enough, and that the issue does not have a significant bearing on anything considered in this paper.

Unfortunately, while the ability for any group of t currently active and connected players to form the Ω update group would make the URSA scheme most reliable and attractive, because of the security vulnerability of this scheme which we describe in this paper, such freedom in choosing Ω would also lead to the fastest key-recovery attack on the resulting scheme. While it is still possible that there exists a smart tweak of the Ω-choosing process which would both slow down in practice the attack we describe here, and would not impact the applicability of the URSA scheme to the "on/off presence, on/off communication links" setting it targets, we believe that what is really needed is a replacement of the URSA proactive RSA scheme with a *provably secure* proactive signature scheme.

IV. An Attack on the URSA Proactive RSA Scheme

A. Overview of the Attack

The goal of the adversary in our attack is to recover the secret-shared private RSA key d in the URSA protocol. Consider an adversary $\mathcal{A}$ who compromises t members. The full attack holds as long as these t members form the "update group" Ω (see section III-C above), and it holds regardless of what indices these players hold. However, for the sake of simplicity in the exposition, we will assume that the adversary corrupts members $M_1, \cdots, M_t$ throughout the lifetime of the scheme, and that these players also always form the Ω update group. We note, however, that our attack does *not* depend on the ability of the adversary to corrupt a *different* set of members every update period. This means that for example, as long as the Ω group is allowed not to change between the updates, the adversary can recover the private RSA key quite quickly if only he corrupts that subset and otherwise follows the protocol so as to avoid detection and revocation of these corrupted members from the group.

Information Leakage in the Signature Protocol. Assume that $\mathcal{A}$ participates in the threshold signature protocol on some message in which the set of participating members G (see section III-B above) is made of all the corrupted members $M_1, \cdots, M_t$ and a single honest member M_p, for some $p \in [t+1, \cdots, 2t]$. (Here too, the attack works for other players M_p, but we fix the above t values of p to simplify the presentation.) Let G_p represent the set of identifiers $\{1, \cdots, t, p\}$ corresponding to the members participating in this run of the threshold signature protocol. By equation (2), the secret key d satisfies the following equation for some integer $\alpha^{(G_p)} \in [0, t]$:

$$d = \sum_{j \in G_p, j \neq p} d_j^{(G_p)} + d_p^{(G_p)} - \alpha^{(G_p)} N \quad \text{(over the integers)}$$

Let us denote $S_p = \sum_{j \in G_p, j \neq p} d_j^{(G_p)}$ (over integers) and $D_p = S_p \pmod N$. Note that since $\mathcal{A}$ knows $ss_1, ss_2, \cdots, ss_t$, he can compute S_p and D_p.

By employing the reconstruction using the *t-bounded offsetting algorithm*, $\mathcal{A}$ learns the value of $\alpha^{(G_p)}$ corresponding to this signing group G_p. Now, note that from values $\alpha^{(G_p)}$ and S_p, the adversary also learns whether the shared secret $d \in [0, \cdots, N-1]$ is less than or greater than D_p. This is because $S_p < \alpha^{(G_p)} N$ if and only if $d < D_p$; and $S_p \geq \alpha^{(G_p)} N$ if and only if $d \geq D_p$.

Utilizing the Information Leakage to Recover the Key. At first sight, the information of whether the secret RSA key d is left or right of some value D_p in the $[0, \cdots, N-1]$ range seems to provide only information on the few most significant bits of d. However, recall that over the lifespan of the system, the members update their secret shares by performing the proactive share update procedure. As we will see below, it turns out that during this procedure, as long as the "update group" Ω is formed by the corrupted players $\{M_1, \cdots, M_t\}$, the adversary can choose the values of his new shares $ss_1, ss_2, \cdots, ss_t$, which gives him complete freedom in specifying the resulting values D_p, for $p = t+1, \cdots, 2t$, to be any values that he wants (we describe this process in subsections IV-B and IV-C below). Since in any subsequent run of the threshold signature protocol involving members $M_1, \cdots, M_t, M_p$ the adversary learns whether the secret d lies to the left or to the right of the corresponding value D_p (for $p = t+1, \cdots, 2t$), the adversary can learn most about d if the chosen values $D_{t+1}, \cdots, D_{2t}$ divide the range $[0, N-1]$ into $t+1$ equally spaced intervals $\{[0, D_{t+1}-1], [D_{t+1}, D_{t+2}-1], \cdots, [D_{2t}, N-1]\}$.

In this case, $\mathcal{A}$ learns from t instances of the threshold signature protocol, for t different values $p = t+1, \cdots, 2t$, whether d lies to the left or to the right of each of these D_p's. Consequently $\mathcal{A}$ shrinks the search interval for the secret d from $[0, N-1]$ to some interval $[D_p, D_{p+1}-1]$ which is smaller than the original interval by the factor of $t+1$. If the adversary repeats this attack recursively, then with every share update protocol his search range narrows by the factor of $t+1$. This is equivalent to saying that in every update interval, the adversary learns the $lg(t+1)$ new most significant bits (MSBs) of the secret d. Therefore, this search procedure will end and the secret key d will be completely recovered after $\lceil \frac{|N|}{lg(t+1)} \rceil$ share update rounds. We refer to this search procedure as a *"(t+1)-ary search"*.

Example with t=1: Consider a simple example with $t = 1$. Assume that the adversary $\mathcal{A}$ compromises member M_1. Assume also that M_1 collaborates with member M_2 in a threshold signature protocol. The signing group is therefore $G_2 = \{1, 2\}$, which yields the following equation:

$$d = d_1^{(G_2)} + d_2^{(G_2)} - \alpha^{(G_2)} N$$

Here $S_2 = D_2 = d_1^{(G_2)}$. $\mathcal{A}$ now employs the *t-bounded offsetting algorithm* and learns the value of $\alpha^{(G_2)}$. If $\alpha^{(G_2)} = 0$, $\mathcal{A}$ learns that $d \geq d_1^{(G_2)} \pmod N$; otherwise if $\alpha^{(G_2)} = 1$, he learns that $d < d_1^{(G_2)} \pmod N$. Assuming that in the share update procedure $\mathcal{A}$ can pick his secret share ss_1 so that the resulting $D_2 = d_1^{(G_2)}$ is whatever $\mathcal{A}$ wants, $\mathcal{A}$ can set $D_2 = \lceil \frac{N-1}{2} \rceil$. Then, with every share update round, $\mathcal{A}$ halves the search interval, and thus he performs a *binary search* which recovers the secret d completely in $lg(N) [= \frac{|N|}{lg(2)}]$ rounds.

Attack Speed-ups: Since in many cases the RSA secret key d can be efficiently recovered once some number of the most significant bits are recovered, the number of rounds in the attack can be further reduced. Moreover, for the commonly used small values of e, like $e = 3$, 17, or 65537, the attack can be sped-up by the factor of two because the first few MSB bits of d enable $\mathcal{A}$ to efficiently compute the first half of the MSBs of d. We describe such speed-up mechanisms in subsection IV-D below.

B. Optimal Choice of New Secret Shares

In each share update protocol the adversary's goal is to set the t values $D_{t+1}, \cdots, D_{2t}$ which will hold in the subsequent update period in a manner described above. In order to do that, the adversary first solves for the optimal new secret shares $ss_1, ss_2, \cdots, ss_t$ which would result in values $D_{t+1}, \cdots, D_{2t}$ he desires, using the following system of t modular linear equations:

$$ss_1 l_1^{(G_{t+1})} + ss_2 l_2^{(G_{t+1})} + \cdots + ss_t l_t^{(G_{t+1})} = D_{t+1} \pmod N$$
$$ss_1 l_1^{(G_{t+2})} + ss_2 l_2^{(G_{t+2})} + \cdots + ss_t l_t^{(G_{t+2})} = D_{t+2} \pmod N$$
$$\cdots\cdots$$
$$\cdots\cdots$$
$$ss_1 l_1^{(G_{2t})} + ss_2 l_2^{(G_{2t})} + \cdots + ss_t l_t^{(G_{2t})} = D_{2t} \pmod N$$

These equations are linearly independent as the following matrix $\mathbb{L}$ is invertible:

$$\mathbb{L} = \begin{pmatrix} l_1^{(G_{t+1})} & l_2^{(G_{t+1})} & \cdots & l_t^{(G_{t+1})} \\ l_1^{(G_{t+2})} & l_2^{(G_{t+2})} & \cdots & l_t^{(G_{t+2})} \\ \cdots & & & \\ l_1^{(G_{2t})} & l_2^{(G_{2t})} & \cdots & l_t^{(G_{2t})} \end{pmatrix} =$$
$$(-1)^t \prod_{i=1}^t \left(\frac{\prod_{j=1,j\neq i}^t (-j)}{\prod_{j=1,j\neq i}^t (i-j)}\right) \begin{pmatrix} \frac{1}{t} & \frac{1}{t-1} & \cdots & 1 \\ \frac{1}{t+1} & \frac{1}{t} & \cdots & \frac{1}{2} \\ \cdots & & & \\ \frac{1}{2t-1} & \frac{1}{2t-2} & \cdots & \frac{1}{t} \end{pmatrix}$$

Thus, by inverting the above matrix the adversary can compute the optimal secret share values $ss_1, \cdots, ss_t$ he needs in the next update round, in order to achieve his optimal values $S_{t+1}, \cdots, S_{2t}$ in the signature protocols performed in that update round. The adversary is now left with the task of forcing the proactive update protocol to actually arrive at these optimal secret shares $ss_1, \cdots, ss_t$ for the corrupted members $M_1, \cdots, M_t$.

C. Adversarial Behavior in the Update

We show that as long as the adversary controls the players in the update group Ω, the adversary can easily influence the proactive share update protocol to arrive at the optimal shares $ss_1, \cdots, ss_n$ he computed above. In the case of larger Ω groups, the attack succeeds as long as the adversary corrupts t members in Ω, and as long as some of these members can "speak last" in the first phase of the URSA share update protocol (see section III-C).

Let us describe the attack assuming the most general case that $|\Omega| \geq t$. Let $B \subseteq \Omega$ denotes the subset of t members corrupted by the adversary, and let $M_b \in B$ be the corrupted member who "speaks last" in the first phase of the share update protocol. Since there is no established sequence or order in which the members in Ω take part in the secret share update procedure, the adversary can wait until each member in Ω except of M_b distributes their shares of the random update polynomials δ_j, $j \in \Omega \backslash \{M_b\}$, before distributing the shares of his polynomial $\delta_b(z)$ as the last one. (If the order is somehow fixed, although it's not clear how it could be without heavy performance penalty for the protocol, the adversary would still win assuming that he corrupts the player who is entitled to speak last.) Recall that the update polynomial is equal to

$$\delta(z) = \sum_{j \in \Omega \backslash \{M_b\}} \delta_j(z) + \delta_b(z) \pmod N$$

and that the new shares of each members are computed as $ss_i = ss'_i + \delta(i)$ where ss'_i is the current share of M_i.

To fix the resulting shares of the corrupted members to come out as the optimal values $ss_1, \cdots, ss_t$ specified above, member M_b chooses his partial update polynomial $\delta_b(z)$ in such a way that the resulting update polynomial $\delta(z)$ satisfies $\delta(i) = ss_i - ss'_i \pmod N$ for $i = 1, \cdots, t$. To do that, M_b sets values $\delta_b(i)$ for $i = 1, \cdots, t$ as

$$\delta_b(i) = ss_i - ss'_i - \sum_{j \in G \backslash \{M_b\}} \delta_j(i) \pmod N$$

The M_b player then interpolates these values to recover the $\delta_b(z)$ update polynomial he should use.

Importantly, note that this adversarial behavior is indistinguishable to outside observers from prescribed behavior an honest player exhibits in the protocol. The attack succeeds if M_b picks his partial update protocol in the above way instead of the prescribed way of picking this polynomial at random, but the difference cannot be observed by the honest players, and thus this attack would be undetected.

D. Speeding-up the Attack

By following the above attack procedure, in every update round the adversary $\mathcal{A}$ learns new $lg(t+1)$ most significant bits (MSBs) of d. Assuming that $\mathcal{A}$ needs to discover all the $|N|$-bits of the RSA secret key d, $\mathcal{A}$ needs $\lceil \frac{|N|}{lg(t+1)} \rceil$ update rounds, and t signature protocol instances within each update interval as described above, to complete the attack. However, there are several ways in which $\mathcal{A}$ can speed up this search. First, we can assume that at least the last 40-bits of the secret d can be obtained by a brute-force search once all the other bits are found, because the candidate d can be efficiently tested given the public key (e, N). This reduces the number of rounds in the attack to $\lceil \frac{1}{lg(t+1)}(|N| - 40) \rceil$. Second, we can speed up this search by making a simple observation about half of the MSBs of d for small e values, and by utilizing several known results regarding the security of the RSA cryptosystem under partial key exposure [4], [2]. Below we explain the speed up for small e's, and we list the other applicable results and explain how they speed up our search algorithm. The graph in Figure (1) summarizes this discussion by showing the number of rounds required in the attack w.r.t the range of the public exponent e for 1024-bit RSA modulus N, taking as an example a threshold value $t = 7$.

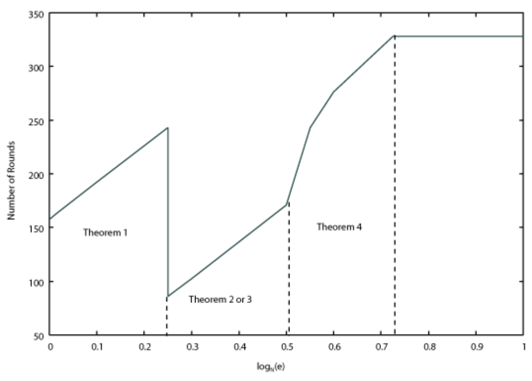

Fig. 1. Currently required # of proactive update rounds to recover d for a given value of $log_N(e)$, assuming $|N| = 1024, t = 7$.

Theorem 1: Given only the first $log_2(e)$ MSBs of d, the first half of MSBs of d can be efficiently computed.

Proof: Note that $ed = 1 \pmod{\phi(N)}$ implies that $d = 1/e(1+k\phi(N))$ for some integer $k = 1, \cdots, e-1$. Therefore, since $N - \phi(N) < \sqrt{N}$, it follows that $0 \le \hat{d}_k - d < \sqrt{N}$ for $\hat{d}_k = \lfloor 1/e(1+kN) \rfloor$ for one of the $e-1$ choices of k. Note that the $\hat{d}_k$ values can be publicly computed, and note that $\hat{d}_{k+1} - \hat{d}_k \approx N/e$ for every k, and therefore that the $log_2(e)$ MSBs of d determine the appropriate k (and $\hat{d}_k$) values, and therefore, since $|\hat{d}_k - d| < \sqrt{N}$ for that k, they also determine the 512 MSBs of d. ■

The import of the above observation for our attack is very simple: Since the above choices of the $e-1$ values $\hat{d}_1, \cdots, \hat{d}_{e-1}$, are neatly spread in the $[0, N-1]$ interval in distances of N/e apart from each other, in our attack based on the $(t+1)$-ary search, the adversary can identify the appropriate $\hat{d}_k$ (and k) value, and thus recover the first $|N|/2$ MSBs of d by the above theorem, after just $\lceil \frac{lg(e)}{lg(t+1)} \rceil$ rounds of the share update protocol.

Therefore, for small e's, the attack requires only

$$r > \frac{1}{lg(t+1)} \left(lg(e) + \frac{|N|}{2} - 40 \right) \tag{3}$$

rounds of share updates for the adversary to learn the whole secret d. This means that the current implementation of URSA which uses the well-known value $e = 65537$ [25] can be attacked in just 163 update rounds for a modest threshold of $t = 7$.

For larger values of e, the results of [4], [2] on the RSA key security with partial key exposure imply the following speed-ups in our RSA key recovery attack:

Theorem 2: [4] If e is a prime in the range $[2^m, 2^{m+1}]$, with $\frac{|N|}{4} \le m \le \frac{|N|}{2}$, then given m MSBs of d, there is a polynomial time algorithm to compute d.

Theorem 3: [4] If e is in range $[2^m, 2^{m+1}]$ and is a product of at most r primes, with $\frac{|N|}{4} \le m \le \frac{|N|}{2}$, then given the factorization of e and m MSBs of d, there is a polynomial time algorithm to compute all of d.

If e meets either of the above criteria, the number of rounds r required in our attack reduces to within the range

$$\lceil \frac{|N|}{4lg(t+1)} \rceil \le r \le \lceil \frac{|N|}{2lg(t+1)} \rceil$$

Theorem 4: [2] If e is in range $[N^{0.5}, N^{0.725}]$, then the number of MSBs needed to completely recover d is given by $\frac{|N|}{8}(3 + 2\alpha + \sqrt{36\alpha^2 + 12\alpha - 15})$ where $\alpha = log_N(e)$.

If e meets the above criteria, the number of rounds required to recover the secret key d is given by $\lceil \frac{|N|}{8lg(t+1)}(3 + 2\alpha + \sqrt{36\alpha^2 + 12\alpha - 15}) \rceil$

V. Discussion

While the above results for general e values are interesting, in practice people want to use RSA with small e values, like $e = 3$, 17, or 65537 (all of which are prime numbers of the form $2^i + 1$ for a small value of i, which makes the exponentiation $s^e \pmod{N}$ involved in the RSA signature verification take only $i+1$ modular multiplications), and for these values our attack holds if (1) t members of the update group Ω are corrupted and one of the corrupted player speaks last, (2) if in every update interval some t chosen honest players M_p are coaxed into participating in a signature protocol (note that the adversary's attack does not depend on what message is used in this protocol), and (3) if the system lasts for $r = 163$ update rounds, which given the twice a day rate of updates gives only two months.

We think that these are quite reasonable assumptions. There's certainly nothing in the adversarial model of a proactive signature scheme and of the URSA group access control scheme, which would disallow the adversary from satisfying each of the above criteria.

As we discussed in the last subsection of section III-C, the attack can probably be slowed down in if some modifications are employed in the process of choosing the "update group" Ω, to make it harder for the adversary to corrupt this group, or, equivalently, to make it harder for the adversary to have whatever players he does corrupt be chosen as the Ω group. However, summing up the discussion from that section, it is not clear how to modify the protocol in order to make the adversary's success significantly harder, and at the same time not to severely limit the applicability of the URSA scheme.

Perhaps more importantly, the attack we exhibit shows that even if the adversary does not satisfy all the above criteria, the adversary still learns meaningful information about the RSA private key d, which makes the security of the resulting system doubtful. For example, as we discussed in the introduction, if the adversary successfully participates in just $r = 34$ instead of $r = 163$ update rounds (for 1024-bit N, $e = 65537$ and $t = 7$), the adversary will learn 600 most significant bits of d. Given the steady progress in the ability to recover the full d from partial knowledge [4], [2], there is little hope that such partial information can be shown not to weaken the RSA system.

Finally, our attack uses only very specific *part* of the information leaked in the URSA threshold RSA signature protocol. We showed that if the adversary corrupts t of the signing $t+1$ players, and if the remaining $(t+1)$-th player M_p can be known beforehand, the leaked information is equivalent to whether the shared secret d lies to the left or to the right of some value D_p which can be computed and in fact controlled by the adversary. However, information about d leaks also if (1) other players M_p, $p \notin [t+1, \cdots, 2t]$ participate in the threshold signature protocol, and if (2) less than t corrupted players participate in this protocol. The information revealed about d in these cases is more complicated than the $d <> D_p$ information we used in our attack, but it is nevertheless easy to define as well, and it can very well be used to either speed up the attack we propose even more, or to extend it to adversaries who (1) corrupt less than t of the players participating in the threshold signature protocol, or (2) corrupt less than t players in the update group Ω, or (3) fail to predict which honest players will participate in the threshold signature protocol.

References

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik. Exploring robustness in group key agreement. In *ICDCS*, 2001.

[2] J. Blomer and A. May. New Partial Key Exposure Attacks on RSA. In *CRYPTO '03*, 2003.

[3] A. Boldyreva. Efficient Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-group Signature Scheme. In *Practice and Theory in Public Key Cryptography*, 2003.

[4] D. Boneh, G. Durfee, and Y. Frankel. An Attack on RSA Given a Small Fraction of the Private Key Bits. In *ASIACRYPT'98*, 1998.

[5] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *ASIACRYPT'01*, 2001.

[6] F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 431–444, 2000.

[7] F. Boudot and J. Traor. Efficient Publicly Verifiable Secret Sharing Schemes with Fast or Delayed Recovery. In *International Conference on Information and Communication Security (ICICS)*, 1999.

[8] J. Camenisch and M. Michels. Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes. In *EUROCRYPT'99*, 1999.

[9] J. Camenisch and M. Michels. Separability and Efficiency for Generic Group Signature Schemes. In *CRYPTO*, 1999.

[10] A. Chan, Y. Frankel, and Y. Tsiounis. Easy Come - Easy Go Divisible Cash. In *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 561–575, 1998.

[11] I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *ASIACRYPT'02*, volume 2501 of *LNCS*, pages 125–142. Springer, 2002.

[12] Y. Desmedt. Society and Group Oriented Cryptosystems. In *CRYPTO '87*, number 293 in LNCS, pages 120–127, 1987.

[13] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *CRYPTO '89*, volume 435 of *LNCS*, pages 307–315, 1990.

[14] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal-Resilience Proactive Public-Key Cryptosystems. In *38th Symposium on Foundations of Computer Science (FOCS)*, pages 384–393, 1997.

[15] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Crypto'97*, volume 1294 of *LNCS*, pages 440–454, 1997.

[16] E. Fujisaki and T. Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In *CRYPTO '97*, 1997.

[17] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. In *EUROCRYPT '96*, 1996.

[18] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. *Information and Computation*, 164:54–84, 1999.

[19] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Public Key and Signature Systems. In *ACM Conference on Computers and Communication Security*, 1997.

[20] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing, Or How To Cope With Perpetual Leakage. In *CRYPTO '95*, volume 963 of *LNCS*, pages 339–352, 1995.

[21] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Conference on Mobile Computing and Networking (Mobicom 2002)*, 2002.

[22] S. Jarecki and N. Saxena. Further Simplifications in Proactive RSA Signatures. In *Theory of Cryptography Conference (TCC'05)*, 2005.

[23] S. Jarecki, N. Saxena, and J. H. Yi. An Attack on the Proactive RSA Signature Scheme in the URSA Ad Hoc Network Access Control Protocol. In *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 1–9, October 2004.

[24] J. Kong, H. Luo, K. Xu, D. L. Gu, M. Gerla, and S. Lu. Adaptive Security for Multi-level Ad-hoc Networks. In *Journal of Wireless Communications and Mobile Computing (WCMC)*, volume 2, 2002.

[25] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for MANET. In *IEEE 9th International Conference on Network Protocols (ICNP)*, pages 251–260, 2001.

[26] S. Lu. Comments on Recent Advances in Cryptoanalysis of URSA. A draft communicated to the authors by email by Songwu Lu in 2004.

[27] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang. URSA: Ubiquitous and Robust Access Control for Mobile Ad Hoc Networks. In *IEEE/ACM Transactions on Networking (ToN)*, December 2004.

[28] H. Luo and S. Lu. Ubiquitous and Robust Authentication Services for Ad Hoc Wireless Networks. Technical Report TR-200030, Dept. of Computer Science, UCLA, 2000. Available online at http://citeseer.ist.psu.edu/luo00ubiquitous.html.

[29] H. Luo, P. Zerfos, J. Kong, S. Lu, and L. Zhang. Self-securing Ad Hoc Wireless Networks. In *Seventh IEEE Symposium on Computers and Communications (ISCC '02)*, 2002.

[30] D. Micciancio and E. Petrank. Simulatable Commitments and Efficient Concurrent Zero-Knowledge. In *EUROCRYPT*, 2003.

[31] M. Narasimha, G. Tsudik, and J. H. Yi. On the Utility of Distributed Cryptography in P2P and MANETs: The Case of Membership Control. In *International Conference on Network Protocol (ICNP)*, 2003.

[32] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2002.

[33] T. Pedersen. Non-interactive and Information-theoretic Secure Verifiable Secret Sharing. In *Crypto 91*, 1991.

[34] T. Rabin. A Simplified Approach to Threshold and Proactive RSA. In *CRYPTO '98*, volume 1462 of *LNCS*, pages 89 – 104, 1998.

[35] N. Saxena, G. Tsudik, and J. H. Yi. Admission Control in Peer-to-Peer: Design and Performance Evaluation. In *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 104–114, October 2003.

[36] N. Saxena, G. Tsudik, and J. H. Yi. Identity-based Access Control for Ad-Hoc Groups. In *International Conference on Information Security and Cryptology (ICISC)*, December 2004.

[37] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[38] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Trans. Parallel Distrib. Syst.*, 11(8), 2000.

[39] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6), 1999.

[40] L. Zhou, F. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *Transactions on Computer Systems*, 2002.

Appendix

A. Robustness Fix to the URSA Proactive RSA Scheme

A trusted dealer shares the RSA private key d by providing random shares $ss_i, ss'_i \in \mathbb{Z}_q$ (for a large prime $q \;\; > N$) to a member M_i using Shamir's polynomial secret sharing [37]. The dealer also broadcasts a commitment to the polynomial (computed mod p, where $q|p-1$) as in the Pedersen's verifiable secret sharing [33].

Each member M_i computes the partial signature $s_i = m^{d_i} \pmod N$, where $d_i = ss_j \; l_j \pmod q$, and l_j denotes the Lagrange coefficient for a given set of $t+1$ signing members. To prove correctness of its partial signature, each M_i proves in zero-knowledge that there is a pair of integers (d_i, d'_i) s.t.

$$w_{i0} = g^{d_i} h^{d'_i} \bmod p \quad , \quad s_i = m^{d_i} \bmod N \quad , \quad 0 \leq d_i < q$$

Here w_{i0} denotes the Pedersen's commitment to M_i's secrets broadcast by the dealer during the sharing phase. It is crucial that the range of d_i is checked because otherwise player M_i can submit its partial signature as $m^{d'_i} \bmod N$ where $d'_i = d_i + kq$ for some k. An efficient zero-knowledge proof system for the proof of equality of discrete logarithms in two different groups was given in [7], [9], and the efficient proof that a committed number lies in a given range appeared in [6]. All these proofs are honest verifier zero-knowledge and can be converted either into standard zero-knowledge proof at the expense of 1-2 extra rounds using techniques of [11], [30], or into a non-interactive proof in the random oracle model using the Fiat-Shamir heuristic. We adopt the notation of [8] for representing zero-knowledge proof of knowledge protocols. For example, $ZKPK\{x : R(x)\}$ represents a ZKPK protocol for proving possession of a secret x which satisfies statement $R(x)$. In the protocols to follow, u (≥ 80) and v (≥ 40) are security parameters.

Protocol for proving the correctness of a partial signature: $ZKPK\{d_i, d'_i : w_{i0} = g^{d_i} h^{d'_i} \pmod p \ \wedge \ s_i = m^{d_i} \pmod N \ \wedge \ d_i \in [0, q-1]\}$

The signer (or prover) M_i proves to the verifier the possession of its correct secret share d_i by using the following zero-knowledge proof system. The verifier can either be one of the players or an outsider who has inputs $w_{i0}, g, h, p, s_i, m, N$, and q. All the protocols run in parallel, and failure of these protocols at any stage implies the failure of the whole proof.

1) The verifier follows the setup procedure of the Damgard-Fujisaki-Okamoto commitment scheme [11], e.g. it picks a safe RSA modulus n and two elements G, H in $\mathbb{Z}_n^*$ whose orders are greater than 2. (We refer to [11] for the details of this commitment scheme.) If N is a safe RSA modulus then set $n = N$, $G = (G')^2 \bmod N$, $H = (H')^2 \bmod N$ for random $G', H' \in \mathbb{Z}_n^*$.
2) The prover computes the commitment $C = G^{d_i} H^R \pmod n$, where R is picked randomly from $[0, 2^v(q-1)]$ and uses **Protocol (1)** (see below), by substituting $(x, x'_1, x'_2, g_1, h_1, g_2, h_2, n_1, n_2, w_1, w_2, b, b')$ with $(d_i, R, d'_i, G, H, g, h, n, p, C, w_{i0}, q-1, 2^v(q-1))$, respectively, to execute: $ZKPK\{d_i, R, d'_i : C = G^{d_i} H^R \pmod n \ \wedge \ w_{i0} = g^{d_i} h^{d'_i} \pmod p\}$
3) The prover then uses **Protocol (1)** (see below), by substituting $(x, x'_1, x'_2, g_1, h_1, g_2, h_2, n_1, n_2, w_1, w_2, b, b')$ with $(d_i, R, 0, G, H, m, m, n, N, C, s_i, q-1, 2^v(q-1))$, respectively, to execute: $ZKPK\{d_i, R : C = G^{d_i} H^R \pmod n \ \wedge \ s_i = m^{d_i} \pmod N\}$
4) The prover uses **Protocol (2)** (see below), by substituting (x, x', b) with $(d_i, R, q-1)$, respectively, to execute: $ZKPK\{d_i, R : C = G^{d_i} H^R \pmod n \ \wedge \ d_i \in [0, q-1]\}$

Protocol (1). $ZKPK\{x, x'_1, x'_2 : w_1 = g_1^x h_1^{x'_1} \pmod{n_1} \ \wedge \ w_2 = g_2^x h_2^{x'_2} \pmod{n_2}\}$
Assumption: $x, x'_2 \in [0, b]$ and $x'_1 \in [0, b']$.

This protocol, from [8], [6], is perfectly complete, honest verifier statistical zero-knowledge and sound under the strong RSA assumption [16] with the soundness error 2^{-u+1}, given than (g_1, h_1, n_1) is an instance of the Damgard-Fujisaki-Okamoto commitment scheme [16], [11].

1) The prover picks random $r \in [1, \ldots, 2^{u+v}b - 1]$, $\eta_1 \in [1, \ldots, 2^{u+v}b' - 1]$, $\eta_2 \in [1, \ldots, 2^{u+v}b - 1]$ and computes $W_1 = g_1^r h_1^{\eta_1} \pmod{n_1}$ and $W_2 = g_2^r h^{\eta_2} \pmod{n_2}$. It then sends W_1 and W_2 to the verifier V.
2) The verifier selects a random $c \in [0, \ldots, 2^u - 1]$ and sends it back to the prover.
3) The prover responds with $s = r + cx$ (in $\mathbb{Z}$), $s_1 = \eta_1 + cx'_1$ (in $\mathbb{Z}$) and $s_2 = \eta_2 + cx'_2$ (in $\mathbb{Z}$)
4) The verifier verifies as $g_1^s h_1^{s_1} = W_1 {w_1}^c \pmod{n_1}$ and $g_2^s h_2^{s_2} = W_2 {w_2}^c \pmod{n_2}$.

Protocol (2). $ZKPK\{x, x' : C = G^x H^{x'} \pmod n \ \wedge \ x \in [0, b]\}$
Assumption: $x \in [0, b]$ and $x' \in [0, 2^v b]$.

This protocol (from [6]) is an exact range proof, honest verifier statistical zero-knowledge, complete with a probability greater than $1 - 2^{-v}$, and sound under the strong RSA assumption given that (G, H, n) is an instance of the Damgard-Fujisaki-Okamoto commitment scheme, similarly as in protocol (1).

1) The prover sets $T = 2(u + v + 1) + |b|, X = 2^T x, X' = 2^T x', \beta = 2^{u+v+1}\sqrt{b}$ and $C_T = G^X H^{X'} \pmod n$.
2) The prover uses **Protocol (3)** (see below), by substituting (x, x', com, B, γ) with $(X, X', C_T, 2^T b, 2^{T/2}\beta)$, respectively, to execute the following (note that $X \in [0, 2^T b]$): $ZKPK\{X, X' : C_T = G^X H^{X'} \pmod n \ \wedge \ X \in [-2^{T/2}\beta, 2^T b + 2^{T/2}\beta]\}$
   Proving that $X \in [-2^{T/2}\beta, 2^T b + 2^{T/2}\beta]$ implies that $x \in [0, b]$, since $2^{T/2}\beta < 2^T$.

Protocol (3). $ZKPK\{x, x' : com = G^x H^{x'} \pmod n \wedge x \in [-\gamma, B + \gamma]\}$ (Here $\gamma = 2^{u+v+1}\sqrt{B}$)
Assumption: $x \in [0, B]$ and $x' \in [0, 2^v B]$.

This proof was proposed in [6] and is honest verifier statistical zero-knowledge, complete with a probability greater than $1 - 2^{-v}$, and sound under the strong RSA assumption just like protocol (2).

1) The prover executes $ZKPK\{x, x' : com = G^x H^{x'} \pmod n\}$
2) The prover sets $x_1 = \lfloor\sqrt{x}\rfloor, x_2 = x - x_1^2$, $\hat{x}_1 = \lfloor\sqrt{B - x}\rfloor, \hat{x}_2 = B - x - \hat{x}_1^2$, and chooses randomly r_1, $r_2, \hat{r}_1, \hat{r}_2$ in $[0, 2^v B]$, such that $r_1 + r_2 = x'$ and $\hat{r}_1 + \hat{r}_2 = -x'$.
3) The prover computes new commitments $e_1 = G^{x_1^2} H^{r_1} \pmod n, \hat{e}_1 = G^{\hat{x}_1^2} H^{\hat{r}_1} \pmod n, e_2 = G^{x_2} H^{r_2} \pmod n, \hat{e}_2 = G^{\hat{x}_2} H^{\hat{r}_2} \pmod n$, and sends e_1 and $\hat{e}_1$ to the verifier.
4) The verifier computes $e_2 = com/e_1 \pmod n$ and $\hat{e}_2 = G^B/(com * \hat{e}_1) \pmod n$.
5) The prover uses **Protocol (4)** (see below), by substituting (x, x', com_{sq}) with (x_1, r_1, e_1) and then with $(\hat{x}_1, \hat{r}_1, \hat{e}_1)$, to execute the following: $ZKPK\{x_1 : e_1 = G^{x_1^2} H^{r_1} \pmod n\}$
   $ZKPK\{\hat{x}_1 : \hat{e}_1 = G^{\hat{x}_1^2} H^{\hat{r}_1} \pmod n\}$ This proves that e_1 and $\hat{e}_1$ hide a square.
6) The prover uses **Protocol (5)** (see below), by substituting (x, x', com_2, B_1) with $(x_2, r_2, e_2, 2\sqrt{B})$, respectively and then with $(\hat{x}_2, \hat{r}_2, \hat{e}_2, 2\sqrt{B})$, respectively, to execute the following (note that x_2 and $\hat{x}_2 \in [0, 2\sqrt{B}]$): $ZKPK\{x_2 : e_2 = G^{x_2} H^{r_2} \pmod n \ \wedge \ x_2 \in [-\gamma, \gamma]\}$, and $ZKPK\{\hat{x}_2 : \hat{e}_2 = G^{\hat{x}_2} H^{\hat{r}_2} \pmod n \ \wedge \ \hat{x}_2 \in [-\gamma, \gamma]\}$. This proves that e_2 and $\hat{e}_2$ hide numbers belonging to $[-\gamma, \gamma]$.
   Steps 2, 5 and 6 above, imply that $x \in [-\gamma, B + \gamma]$.

Protocol (4). $ZKPK\{x, x' : com_{sq} = G^{x^2} H^{x'} \pmod n\}$

This protocol appeared in [16], generalized in [11] and proves that a committed number is a square. It is honest verifier statistical zero-knowledge, perfectly complete, and sound under the strong RSA assumption just like protocol (2).

Protocol (5). $ZKPK\{x, x' : com_2 = G^x H^{x'} \pmod n \ \wedge \ x \in [-2^{u+v}B_1, 2^{u+v}B_1]\}$
Assumption: $x \in [0, B_1]$, and $x' \in [0, 2^v B_1]$.

This proof, proposed in [10], allows a prover to prove the possession of a discrete logarithm x lying in the range $[-2^{u+v}B_1, 2^{u+v}B_1]$ given x which belongs to a smaller interval $[0, B_1]$. Using the commitment scheme of [16], [11], this proof is honest verifier statistical zero-knowledge, complete with a probability greater than $1 - 2^{-v}$, and sound under the strong RSA assumption with soundness error 2^{-u+1}.

Stanislaw Jarecki received the B.S. and Ph.D. degrees in computer science from the Massachusetts Institute of Technology, Cambridge, in 1996 and 2001, respectively. He is an Associate Professor of Computer Science at the University of California, Irvine. His research interests are cryptography, security, and distributed algorithms.

Nitesh Saxena is an Assistant Professor in the Department of Computer Science and Engineering at Polytechnic Institute of New York University (formerly Polytechnic University). He works in the areas of computer and network security, and applied cryptography. Nitesh obtained his Ph.D in Information and Computer Science from UC Irvine. He holds an M.S. in Computer Science from UC Santa Barbara, and a Bachelors degree in Mathematics and Computing from the Indian Institute of Technology, Kharagpur, India. Niteshs Ph.D. dissertation on "Decentralized Security Services" has been nominated for the ACM Dissertation Award 2006. He is the recipient of the Best Student Paper Award at the Applied Cryptography and Network Security (ACNS) conference 2006.