Proactive RSA Signatures with Non-Interactive Signing

Stanislaw Jarecki and Josh Olsen

School of Information and Computer Science
University of California, Irvine

Talk Outline

- Threshold Signatures and Proactive Signatures
  - Model and Motivation
  - Importance of Proactive Security
  - Importance of Non-Interactive Signing
- Ingredients of our Protocol:
  - Threshold RSA Signature of Shoup
  - Proactive RSA Signature of Rabin
- Our Protocol: Proactive RSA with Non-interactive Signing
- Extensions and Open Questions

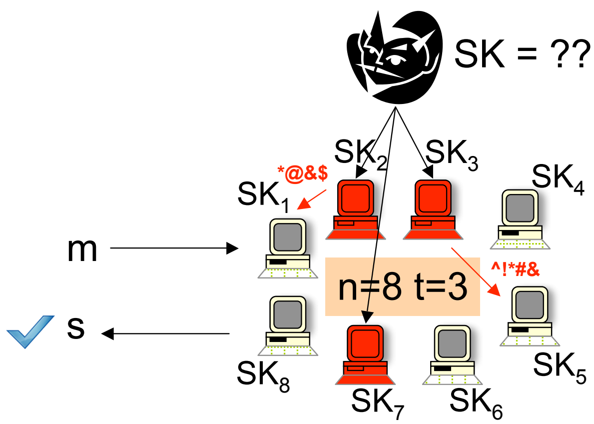
Threshold Signatures: Main Idea

Share the secret key among n players, $SK \rightarrow (SK_1, \dots, SK_n)$, s.t. we can *securely tolerate* corruption of t out of n players.

i.e. if *an adversary* corrupts at most t out of n players

[Security:] he does not learn anything about the key SK (and cannot forge signatures)

[Robustness:] he cannot prevent the computation of a correct signature by the remaining n-t players



Applications: Fault-Resistance

1. Roots of Trust
- Certification Authority
- Time-stamping

2. Secure Services
- Access Control
- Storage

3. Decentralized Groups

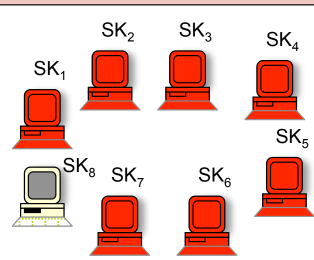
Beyond Threshold Cryptosystems

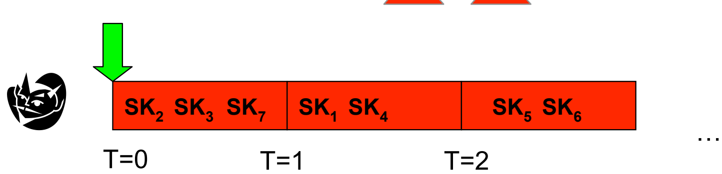
Fundamental Limit of Threshold Cryptosystems:
What if the adversary eventually corrupts more than t players?

Eventual corruption of all players is easier than you think:
- inevitable eventual breakdown
- periodical service / upgrades

Stronger Adversary: *Mobile* Adversary, who corrupts up to t players in every fixed time interval

Mobile Adversary eventually compromises any threshold cryptosystem…





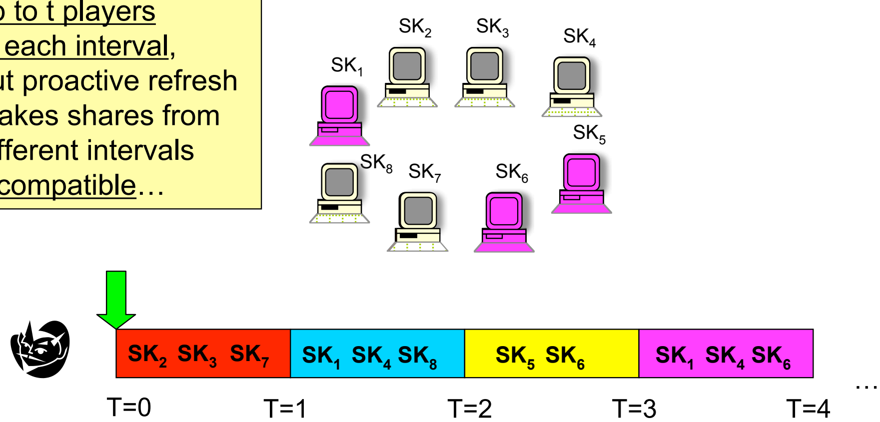
Solution: Proactive Security

Main Idea: Refresh the sharing of the key between each interval

=> Secrets learned in one interval are useless in another

=> System tolerates up to t corruptions in *each time interval*

Adversary corrupts up to t players in each interval, but proactive refresh makes shares from different intervals incompatible…



Previous Work on Proactive and Threshold RSA

Threshold RSA: *signing protocol is always fast (non-interactive)*

- Desmedt-Frankel'90: heuristic security
- DDFY'94: secure, but O(n)-sized shares
- FGY'96, GJKR'96: extension to malicious security
- Shoup'00: O(1)-sized shares, "safe" RSA modulus
- DK'01, DD'04: larger class of RSA public keys

Proactive RSA:

- FGM'97a: combinatorial scheme
- FGM'97b: polynomial shares, re-sharing per signature
- Rabin'98: simplification of FGM'97b, *interactive signing*
- JS'05: reduced share sizes

Adaptive Security in Proactive RSA:

- CGJKR'99, JL'01, FMY'01, ADN'06

- Best *Threshold* RSA has non-interactive signing
- Best *Proactive* RSA has interactive (2 stage) signing

Problems with Interactive Signing of Rabin'98 (and JS'04):

- Signing in 1st round requires presence of all n players
  - ⇒ Protocol takes 2-rounds if one player is missing / slow
- If player is missing, his share is publicly reconstructed in 2nd round
  - ⇒ Communication faults are equated with malicious faults
  - ⇒ Much worse security in practice, where communication faults are much easier to induce than corruptions
  - ⇒ Unusable for networks where partitions are common
    - e.g. Peer to peer, MANETs, sensors, and others…

[Almansa, Damgard, Nielsen '06: 2-round, no public reconstruction*]
(*) Remains interactive, but achieves adaptive security

	Interactive Signing	Non-Interactive Signing
Proactive Security	Rabin '98	This Work
Only Threshold	N/A	Shoup '00

← (This Work):
- One-round signing, needs only t of n players (costs almost as Shoup'00)
- No public share reconstruction
- Efficient proactive update (as in Rabin'98)

Threshold RSA: [Shoup'00]

$N = p*q$, $\varphi(N)=(p-1)(q-1)$
$e*d = 1 \bmod \varphi(N)$
PK = e, SK = d
Sign: $s \leftarrow m^d \bmod N$
Ver.: $m = s^e$ [$= m^{d*e}$] mod N

Given RSA instance (N,e,d)
Shamir's secret sharing modulo $\varphi(N)$:
- pick t-degree polynomial f s.t. $d = f(0) \bmod \varphi(N)$
- player P_i gets a "share" $d_i = f(i) \bmod \varphi(N)$

[Security: f is a t-degree poly. → f(0) is independent from any t values of f]

- Recall polynomial interpolation (over integers):
  For any set G of t+1 indexes i, there are (rational) constants c_i s.t.

$$f(0) = \sum_{i \in G} c_i \cdot f(i)$$

Non-Interactive Signing!

- Each P_i outputs $s_i = m^{d_i} \bmod N$
- Compute RSA signature from s_i's of any t+1 (honest) players:

$$s = \prod_{i \in G} (s_i)^{c_i} \bmod N \quad [= \prod m^{c_i \cdot d_i} = m^{\sum c_i \cdot f(i)} = m^{d}]$$

Problem: Lagrange Interpolation c_i constants are not integers!
$c_i = (\Pi_{j \text{ in } G}\, j) / (\Pi_{j \text{ in } G}\, j-i)$
Exponentiation to fractional exponent = computing roots mod N: Hard under RSA!

Threshold RSA: [Shoup'00]

$N = p*q$, $\varphi(N)=(p-1)(q-1)$
$e*d = 1 \bmod \varphi(N)$
$PK = e$, $SK = d$
Sign: $s \leftarrow m^d \bmod N$
Ver.: $m = s^e$ [$= m^{d*e}$] $\bmod N$

Given RSA instance (N,e,d)
Shamir's secret sharing modulo $\varphi(N)$:

- pick t-degree polynomial f s.t. $d = f(0) \bmod \varphi(N)$
- player P_i gets a "share" $d_i = f(i) \bmod \varphi(N)$

- Recall polynomial interpolation (over integers):
  For any set G of t+1 indexes i, there are **integer**

$$L*f(0) = \sum_{i \in G} c_i \cdot f(i)$$

- Each P_i outputs $s_i = m^{d_i} \bmod N$
- Compute RSA signature from s_i's of any t+1 (ho

$$\hat{s} = \prod_{i \in G} (s_i)^{c_i} \bmod N \quad [= \prod m^{c_i \cdot d_i} = m^{\sum c_i \cdot f(i)} = m^{L*d}]$$

Compute: $m^{Ld} \rightarrow m^d$

If gcd(e,L)=1, use Euclidean Algorithm to find a,b s.t. ae + bL= 1

$s = m^a * \hat{s}^b$

Check:
$s^e = m^{ae} * (m^{Ld})^{be} =$
$= m^{ae + Lb} = m$

Problem: Lagrange Interpolation c_i constants are not integers!
$c_i = (\Pi_{j \text{ in } G}\, j) / (\Pi_{j \text{ in } G}\, j-i) * L$, where $L=n!$ => c_i's are integers now!
Exponentiation to fractional exponent = computing roots mod N: Hard under RSA!

Problem #2:
Not clear how to argue that m^{d_i}'s reveal no additional information about d than m^d...
How to simulate(m^d, $d_1, \ldots, d_t$) $\rightarrow m^{d_i}$?

- $d_i = c_0 d + c_1 d_1 + \ldots + c_t d_t$ for Lagrange coefs. c_j's
- $m^{d_i} = (m^d)^{c_0}\, m^{(c_1 d_1 + \ldots + c_t d_t)}$
- But these exponents also can be fractions…

$N = p*q$, $\varphi(N)=(p-1)(q-1)$
$e*d = 1 \bmod \varphi(N)$
$PK = e$, $SK = d$
Sign: $s \leftarrow m^d \bmod N$
Ver.: $m = s^e$ [$= m^{d*e}$] $\bmod N$

Solution #2:

- Publish $s_i = m^{Ld_i}$
  instead of $s_i = m^{d_i}$
- Simulation:
  $m^{Ld_i} = (m^d)^{Lc_0}\, m^{L(c_1 d_1 + \ldots)}$
- Now $\hat{s} = m^{L*L*d}$, not m^{L*d}
- Euclidean Algorithm($\hat{s}$) $\rightarrow m^d$
  because $\gcd(e,L^2)=1$

- Recall polynomial interpolation (over int
  For any set G of t+1 indexes i, there are

$$L*f(0) = \sum_{i \in G} c_i \cdot f(i)$$

- Each P_i outputs $s_i = m^{d_i} \bmod N$
- Compute RSA signature from s_i's of any t+1 (honest) players:

$$\hat{s} = \prod_{i \in G} (s_i)^{c_i} \bmod N \quad [= \prod m^{c_i \cdot d_i} = m^{\sum c_i \cdot f(i)} = m^{L*d}]$$

Problem: Lagrange Interpolation c_i constants are not integers!
$c_i = (\Pi_{j \text{ in } G}\, j) / (\Pi_{j \text{ in } G}\, j-i) * L$, where $L=n!$ => c_i's are integers now!
Exponentiation to fractional exponent = computing roots mod N: Hard under RSA!

How to “Proactivize” (Shoup’s) Threshold RSA?

Given RSA instance (N,e,d)

Shamir’s secret sharing modulo φ(N):

- pick t-degree polynomial f s.t. $d = f(0) \mod \varphi(N)$
- player P_i gets a “share” $d_i = f(i) \mod \varphi(N)$

- Recall polynomial interpolation (over integers):

  For any set G of t+1 indexes i, there are **integer** constants c_i s.t.

$$L * f(0) = \sum_{i \in G} c_i \cdot f(i)$$

- Each P_i outputs $s_i = m^{L d_i} \mod N$
- Compute RSA signature from s_i’s of any t+1 (honest) players:

$$\hat{s} = \prod_{i \in G} (s_i)^{c_i} \mod N \quad [= \prod m^{c_i \cdot d_i} = m^{\sum c_i \cdot f(i)} = m^{L^2 * d}]$$

How to “Proactivize” (Shoup’s) Threshold RSA?

$N = p \cdot q$, $\varphi(N) = (p-1)(q-1)$

$e \cdot d = 1 \mod \varphi(N)$

PK = e, SK = d

Sign: $s \leftarrow m^d \mod N$

Ver.: $m = s^e$ [$= m^{d \cdot e}$] mod N

Given RSA instance (N,e,d)

Shamir’s secret sharing modulo φ(N):

- pick t-degree polynomial f s.t. $d = f(0) \mod \varphi(N)$
- player P_i gets a “share” $d_i = f(i) \mod \varphi(N)$

Recall: Proactive Refreshment [HJKY’95] (applied to RSA)

- Pick t-degree polynomial δ s.t. $\delta(0) = 0 \mod \varphi(N)$
- Each P_i gets an “update share” $\delta(i) \mod \varphi(N)$
  - P_i re-computes share $d'_i \leftarrow d_i + \delta(i) \mod \varphi(N)$
- Note: $d'_i = f(i) + \delta(i) = f'(i) \mod \varphi(N)$,

  where $f' = f + \delta$ is a t-degree poly. s.t. $f'(0) = f(0) = d \mod \varphi(N)$

Q1: Who picks δ ?

A: Easy! Each P_i picks $\delta^{(i)}$, shares it, and $\delta = \delta^{(1)} + \ldots + \delta^{(n)}$

Q2: How to do share $\delta^{(i)}$ when no one knows the modulus φ(N) ??

A: Not so easy…

… but achieved in [FGM97b] with secret-sharing over integers

Shamir's Secret-Sharing over Integers [FGMY97b]

Given secret d in [0,R]
Pick vector **a** = ($a_1,\ldots,a_t$) of coefficients at random in [0,…,$RtL^2 2^k$]
Define $f(x) = Ld + a_1x + \ldots a_tx^t$
P_i's share: $s_i = f(i)$ [over integers]

Let the set of corrupt players be {1,…,t}
Let **s** = ($s_1,\ldots,s_t$), w = Ld, **w** = (w,…,w)
Note that **s** = **w** + M**a**

$$M = \begin{bmatrix} 1 & 1^2 & \ldots & 1^k \\ 2 & 2^2 & \ldots & 2^k \\ \vdots & \vdots & \ldots & \vdots \\ t & t^2 & \ldots & t^k \end{bmatrix}$$

Security:
Compare distributions of **s** given $\mathbf{w_1}$ and $\mathbf{w_2}$:

$$\mathbf{s} = \mathbf{w}_1 + M\mathbf{a}_1 = \mathbf{w}_2 + M\mathbf{a}_2$$

$$\Rightarrow \quad (\mathbf{w}_1 - \mathbf{w}_2) = M(\mathbf{a}_1 - \mathbf{a}_2)$$

$$\Rightarrow \quad \mathbf{a}_1 = \mathbf{a}_2 + M^{(-1)}(\mathbf{w}_1 - \mathbf{w}_2)$$

Entries of $M^{(-1)}$ are similar to Lagrange coefficients: $\Pi_k (i-k) / (j-k)$

1. Why length? Since $(\mathbf{w}_1 - \mathbf{w}_2) < \delta w < LR$, and highest element in $M^{(-1)}$ is tL, the mask size should be $LR\cdot tL\cdot 2^k$
2. Why Ld? $M^{(-1)}$ has non-integer entries, but denominators divide L=n!

Tal Rabin's Proactive RSA [Rabin98]

- Secret key d shared additively → $(d_1,\ldots,d_n)$ s.t. $d_1+\ldots+d_n = d$ [this is a simplification]
- Each d_i is shared using Shamir's secret-sharing over integers
- Proactive refresh protocol is simple:
  - Each P_i shares d_i additively → $(d_{i1},\ldots,d_{in})$ s.t. $d_{i1} +\ldots+ d_{in} = d_i$
  - Each P_i sends d_{ij} to P_j
  - P_j computes $d_j' \leftarrow d_{1j} + d_{2j} +\ldots + d_{nj}$ and shares it over integers
- Signing is conceptually simple:
  - Each player produces m^{d_i}
  - Missing d_i's are publicly reconstructed from the back-up sharings

5. However, this signing protocol is:
   - interactive (unless all n players are present) and
   - exposes shares (e.g. insecure if network is partitioned)

Our Protocol: Proactive RSA *with Fast Signing*

- Secret key d shared additively → $(d_1,\ldots,d_n)$ s.t. $d_1+\ldots+d_n = d$ [this is a simplification]

- Each d_i is shared using <u>Shamir's secret-sharing over integers</u>

- Proactive refresh protocol is simple:
  - Each P_i shares d_i additively → $(d_{i1},\ldots,d_{in})$ s.t. $d_{i1} +\ldots+ d_{in} = d_i$
  - Each P_i sends d_{ij} to P_j
  - P_j computes $d_j' \leftarrow d_{1j} + d_{2j} +\ldots + d_{nj}$ and shares it over integers

---

- Signing with <u>Shamir's secret-sharing over integers:</u> [FGMY'97b,Rab98]
  - By linearity of Shamir-SS-over-Z:
    - Sharings of $(d_1,\ldots,d_n)$ imply Sharing of $d = d_1+\ldots+ d_n$
    - Shamir-SS over integers → $f(0) = Ld$ (*instead of d*)
  - Signing protocol *similar* to Shoup's: [Shoup'00]
    - Each player produces m^{Ld_i}
    - Interpolation reconstructs $m^{L^3 d}$ (*instead of* $m^{L^2 d}$)
    - Euclidean Algorithm reconstructs m^d

Extensions and Open Problems

Extensions:

- More exact security argument for Secret-sharing over integers
  - Share size reduced to ≤ |N| + sec.par. + 3log(n!)

- Further extension: Getting rid of additive sharing altogether
  - Proactive refresh protocol can be done by only t players
  - Using verifiable encryption it can be done non-interactively

Open Questions:

- Extension to more general RSA moduli N. (Now: safe RSA modulus)

- Extension to e=3. (Now: require gcd(e,n!)=1)

- Removing the n! factor completely
  - This would allow very large groups, e.g. peer-to-peer, MANETs
  - Indexes could be MAC addresses instead of consecutive integers