Threshold Signatures
Part 2: RSA
Bar-Ilan University Winter School on Cryptography

Rosario Gennaro
The first public key signature

RSA signatures

Let $N=pq$ be the product of two primes.

On input a message $M$, we hash it to obtain $m \in \mathbb{Z}_N$ and compute the signature $s = m^d$.
Let's start with additive

$n$-out-of-$n$ RSA signatures

- A dealer generates $N, e, d$ and shares the secret key $d$ among $n$ parties additively
  - Let $[d_1 \ldots d_n]$ be the shares chosen at random in $\mathbb{Z}_{\varphi(N)}$
    - such that $d = d_1 + \ldots + d_n \mod \varphi(N)$
  - To sign players reveal $s_i = m^{d_i} \mod N$
    - Then $s = s_1 \ast \ldots \ast s_n \mod N$

- Why is this secure?
  - Same interpolation in the exponent argument as in the case of dlog schemes
  - The simulator gives random $d_i$ to the adversary
    - given $s$ it can compute the partial signatures of the honest players
  - Random $d_i$ to chosen where? The simulator does not know $\varphi(N)$
    - It chooses them in $\mathbb{Z}_N$
      - Since the uniform distributions in $\mathbb{Z}_{\varphi(N)}$ and $\mathbb{Z}_N$ are indistinguishable
        - When $p \sim q$

Move to threshold

Shamir’s over a ring

- The dealer can share $d$ with Shamir’s
  - Choose a random polynomial $F(x) \in \mathbb{Z}_{\varphi(N)}[X]$ of degree $t$ such that $F(0) = d$
  - Send to player $P_i$ the share $d_i = F(i) \mod \varphi(N)$

- A set $S$ of $t+1$ players cannot recover the secret by polynomial interpolation
  - To compute the Lagrangians they need to invert elements $\mod \varphi(N)$
  - Which is secret and cannot be leaked to the participants

- Remember that $d = \sum_{i \in S} \lambda_{i,S} d_i$
  - where $\lambda_{i,S} = \left( \prod_{j \in S, j \neq i} j \right) / \left( \prod_{j \in S, j \neq i} (j-i) \right) \mod \varphi(N)$
    - which cannot be computed by the players
  - What the players can compute is $(n!)d$ by revealing $(n!)d_i$
    - Since $(n!)\lambda_{i,S}$ is an integer
Threshold RSA First Attempt

t-out-of-n RSA signatures

- A dealer generates $N, e, d$ and shares the secret key $d$ among $n$ parties with Shamir's secret sharing.
  - Let $[d_1, ..., d_n]$ be the shares.
  - To sign players reveal $s_i = m^{d_i} \mod N$.
    - Then $s^{n!} = \prod_{i \in S} s_i^{n!} = m^{d \cdot n!} \mod N$.

- How do we get $s$?
  - Assume that $GCD(e, n!) = 1$ (choose $e > n$).
    - Use Extended Euclidean algorithm to compute $a, b$ such that $a \cdot e + b \cdot n! = 1$.
    - Then by the famous Shamir’s trick:
      - $s = m^d = m^{d(a \cdot e + b \cdot n!)} = m^a \cdot m^{b \cdot n!} = m^d = m^a \cdot s^b \mod N$.

Victor Shoup: Practical Threshold Signatures. EUROCRYPT 2000: 207-220
Threshold RSA

Let’s try to Simulate

Simulator computes the adversary’s *partial signatures* $s_i = m^{d_i}$ and knows $s_0 = s = m^d$

- But cannot interpolate in the exponent the partial signatures of the honest players
  - Since $d_j = \sum_{i \in S} \lambda_{j,i,S} d_i$ then $s_j = \prod_{i \in S} s_i^{\lambda_{j,i,S}}$
  - And the Lagrangians are fractions
- He can however interpolate $s_j^{n!} = \prod_{i \in S} s_i^{n! \cdot \lambda_{j,i,S}}$

Assume the adversary can forge controlling only $t$ players

Simulator gives random $d_i$ to the adversary and plays the role of the honest players

$PK=(N,e)$

SIMULATOR (forging centralized scheme)
### Threshold RSA

**t-out-of-n RSA signatures**

- A *dealer* generates $N, e, d$ and shares the secret key $d$ among $n$ parties with Shamir modular $\varphi(N)$
  - Let $[d_1, ..., d_n]$ be the shares
  - To sign players reveal $s_i = m^{d_i} \cdot n! \mod N$
    - Then $s^z = \prod_{i \in S} s_i^{n!} \cdot \lambda_i, S = m^{d^z} \mod N$
    - Where $z = (n!)^2$
  - We get $s$ via the GCD trick again assuming that $GCD(e, z) = 1$ (choose $e > n$)

---

**Victor Shoup: Practical Threshold Signatures. EUROCRYPT 2000: 207-220**
Threshold RSA

Simulation

Simulator computes the adversary $t$ modified partial signatures $u_i = m^{d_i}$ and knows $u_0 = s = m^d$

- It can interpolate in the exponent the partial signatures of the honest players
  
  - Since $d_j = \sum_{i \in S} \lambda_{j,i,s} d_i$ then $s_j = \prod_{i \in S, s_i \lambda_{j,i,s}}$
  
  - And the Lagrangians are fractions

- $u_j^{n!} = \prod_{i \in S, u_i^{n! \lambda_{j,i,s}}}$

Assume the adversary can forge controlling only $t$ players

Simulator gives random $d_i$ to the adversary and plays the role of the honest players

$PK = (N,e)$

SIMULATOR (forging centralized scheme)
What if the identifiers are big

**Ad-hoc groups**

- In the previous solution the value $n$ is a parameter to the scheme
  - Computation is linear in $n$ (exponentiate to $n!$)
  - Assumes that the identifiers of the players are exactly integers between 1 and $n$
    - $n!$ grows really large if identifiers are random $k$-bit strings

- To sign players reveal $s_i = m^{d_i} \cdot n! \mod N$
- Then $s^z = \prod_{i \in S} s_i^{n!} \cdot \prod_{i \in S} = m^{d^*z} \mod N$

This one can be replaced with $lcm\{(i-j)\}$ for $i,j \in S$<sub><sup>2^k</sup></sub>

---

Ad-Hoc Groups Threshold RSA

Back to the Simulation

Simulator computes the adversary's partial signatures $s_i = m^{d_i}$ and knows $s_0 = s = m^d$.

- To interpolate in the exponent, the partial signatures of the honest players need to compute a $z_A$-root of $m$.
  - Where $z_A$ is the product of all the denominators of the adversary's Lagrangians.

Knowledge of $s'$ allows the simulator to complete the simulation.

Assume the adversary can forge controlling only $t$ players.

Simulator gives random $d_i$ to the adversary and plays the role of the honest players.

If $\text{GCD}(e,z_A) = 1$ conjectured not to help find $e$-roots.
Adding robustness

Dealing with bad partial signatures

- Remember that on message $M$ a player outputs $s_i = m^d_i \mod N$
  - How to detect bad partial signatures?

- **Message Authentication Codes:**
  - For every share $d_i$, the dealer chooses $n$ triplets $(a_{ij}, b_{ij}, c_{ij})$ such that
    - $a_{ij} \times d_i + b_{ij} = c_{ij}$ over the integers
    - With $a_{ij} \in [0..2^{k1}]$ and $b_{ij} \in [0..2^{k2}]$ chosen uniformly at random
    - And sends $c_{ij}$ to player $i$ and $a_{ij}, b_{ij}$ to player $j$

- When player $i$ outputs $s_i = m^d_i \mod N$
  - It sends to player $j$ the value $C_{ij} = m^{c_{ij}} \mod N$
  - Player $j$ accepts $s_i$ if $s_i^{a_{ij}} \times m^{b_{ij}} = C_{ij} \mod N$

---

Adding robustness

Dealing with bad partial signatures

- Remember that on message $M$ a player outputs $s_i = m^d_i \mod N$

- **Zero-Knowledge Proofs:**
  - For every share $d_i$, the dealer publishes $G_i = g^{d_i} \mod N$
  
  - When player $i$ outputs $s_i = m^d_i \mod N$
    - It also sends a ZK proof that $s_i$ and $G_i$ to have the same discrete log with respect to $m$ and $g$
    - It requires restricting $m,g$ to a cyclic subgroup of $\mathbb{Z}_N^*$
      - For safe primes the subgroup of quadratic residues

---

Chaum’s prescience

Equality of discrete log ZK Proofs in groups of prime order

\[ y = g^x \quad s = m^x \]

Public coin: can be made non-interactive via Fiat-Shamir. Proof of knowledge of \( x \)

Private coin (can’t be made non-interactive). Two round HVZK (can be turned into 4-round full Zk)

\[ a = g^r, b = m^r \]

\[ d = r + cx \mod q \]

\[ c = g^a m^b \]

\[ d = c^x \]

\[ y^c = g^d \]

\[ b s^c = m^d \]

D. Chaum, T. P. Pedersen: Wallet Databases with Observers. CRYPTO 1992

D. Chaum, H. Van Antwerpen: Undeniable Signatures. CRYPTO 1989

\[ a, b \in \mathbb{Z}_q \]

\[ r \in \mathbb{Z}_q \]
Composite order
Equality of discrete log ZK Proofs in groups of unknown order

\[ y = g^x \quad s = m^x \]

Choose \( r \) large enough to statistically hide \( x \)

\[ a = g^r, b = m^r \]
\[ c \]
\[ d = r + cx \quad \text{over the integers} \]
\[ \begin{align*}
\text{Condition: } & ay^c = g^d \\
& bs^c = m^d \\
\end{align*} \quad \text{mod } N \]

\[ c = g^a m^b \]
\[ d = c^x \]
\[ d? = y^a s^d \]

---

Wait a minute

DEALER?

- This time removing the dealer is not as easy as in the case of discrete log based schemes
  - The dealer does not just generate a random value
  - It generates an RSA modulus $N$ with its factorization and then the values $e,d$
- To replace the dealer we need to come up with a protocol to do all of the above distributed with the above secrets (the factorization) in shared form
  - While in principle this is obtainable via MPC protocols it is still a difficult task to perform efficiently
    - The bottleneck would be the repeated computation of modular exponentiations in a distributed Miller-Rabin primality test
    - This has been a very active research area
Distributed RSA Generation

Avoiding Miller-Rabin

- Let’s break down the task:
  a. The $n$ parties generate a random number and do a preliminary sieving (to make sure that it is not divided by small primes)
  b. Given two such numbers $p,q$ the parties compute $N=pq$
  c. The parties now distributively test that $N$ is bi-prime (the product of 2 primes)
  d. If the test succeeds the parties compute $e,d$
Distributed RSA Generation

Sieving and Multiplication

a. The n parties generate a random number and do a preliminary sieving (to make sure that it is not divided by small primes)
   - Each party generates random numbers $p_i \in [0...B]$ and $r_i \in [0...B']$
   - Reconstruct $pr$
     - Multiplication of additively shared values
     - And reject $p$ if $pr \equiv 0 \mod a$
       - Where $a$ is the small prime

b. Given two such numbers $p,q$ the parties compute $N=pq$
   - Again this is the multiplication of additively shared values

Distributed RSA Generation

Bi-primality testing

c. The parties now distributively test that $N$ is bi-prime (the product of 2 primes)
   - A very simplified version
   - Remember that $N = pq$ and the parties have additive sharings of $p, q$
   - If $N$ is bi-prime then $\varphi(N)$ is the order of $\mathbb{Z}_N^*$
     - The parties have an additive sharing $\varphi_1, \ldots, \varphi_N$ of $\varphi(N) = N - p - q + 1$
   - Repeat many times:
     - The parties choose a random value $g$ and test if $g^{\varphi(N)} = 1$
     - **Locally** compute $g_i = g^{\varphi_i}$
     - Use a distributed computation to check that $g_1 \cdot \ldots \cdot g_n = 1$
       - Can’t reveal the $g_i$
   - An additional GCD test is also required
Distributed RSA Generation

Inversion over a shared secret

d. The parties now choose $e$ and compute $d = e^{-1} \mod \varphi(N)$
   - This is the “dual” problem of the one we saw yesterday
     - In the DSA scheme we had a public modulus and we had to invert the secret
     - Here we have a public value to invert but a secret modulus
   - The parties have an additive sharing $\varphi_1 \ldots \varphi_N$ of $\varphi(N)$
     - Choose a random value $r_i \in [0 \ldots B]$ and let $r = r_1 + \ldots + r_n$
     - Reveal $a_i = \varphi_i + e r_i$
       - $a = a_1 + \ldots + a_n = \varphi(N) + re$
     - If GCD$(a,e)=1$ then there exists $b,c$ such that $ab + ce = 1$
       - $1 = ab + ce = b\varphi(N) + (br+c)e$
       - $br+c = e^{-1} \mod \varphi(N)$
   - Shares of $d$ can be easily obtained by setting $d_i = br_i$
     - With one party adding $c$ as well

A little detour

Signatures based on Strong-RSA

We have been looking at the basic “hash and sign” RSA signature

- Which are proven in the random oracle model
- There are provably secure schemes based on the Strong-RSA assumption
  - Given \((N, g)\) find \((e, s)\) such that \(s^e = g \mod N\)
- These schemes work as follows:
  - The public key is \((N, g)\) and the secret key is \(\varphi(N)\)
  - a message \(M\) is mapped into an exponent \(m\) and the signature is \(s = g^d \mod N\)
    - where \(d = m^{-1} \mod \varphi(N)\)
  - The pair \((M, s)\) is valid if \(s^m = g \mod N\)

- To make these schemes into threshold ones we need exactly the protocol we showed before
  - Given \(m\) compute a sharing of \(d = m^{-1} \mod \varphi(N)\)
  - Over a distributed \(\varphi(N)\)
The two-party case

The Boneh-Franklin protocol required honest majority and was proven only for the honest but curious adversary setting

- Gilboa showed how to extend it for the 2-party case
- In particular introducing the MtA protocols we discussed yesterday
More Distributed RSA Generation

Many follow up works

There are several applications beyond threshold RSA signatures that could use a distributed generation of RSA moduli

- Many protocols have been presented following the Boneh-Franklin approach with improvements focused on
  - Increasing the rate of sieving to avoid running the bi-primality test too often
  - Reducing communication complexity
    - E.g. use a distributed version of the MtA protocol using a threshold additively homomorphic encryption
    - Since one cannot use Paillier, use lattice-based one instead
  - Adding security against malicious adversary via ZK proofs
    - Using recent advances in SNARKs (sublinear size proofs)
  - We can now generate distributed RSA moduli for 1000’s of parties in a matter of minutes.