June 6: Post-Quantum Crypto

REMINDE: Project!!

But first, review.

Review: Classical Cryptanalysis

Discrete Log: group $G = \langle g \rangle$ of prime order $p$

- Given $g, h \in G$, find $x$
- Idea: Find collision in $F(\alpha, \beta) = g^\alpha h^\beta \in G$

BGS: Time $\tilde{O}(p^2)$, Space $O(p)$
Rolled Rho: Time $\tilde{O}(p^2)$, Space $O(1)$

Shoup: Can't do better "generically" => $\exists$ faster alg for specific $G$ (e.g. GNFS)

Factoring:

- Split $N = pq$
- Trial Division: Time $O(\sqrt{N})$, Space $O(1)$
- Pollard Rho: Time $\tilde{O}(N^{1/4})$, Space $O(1)$
- Dixon: $L_p[\frac{1}{2}, \frac{1}{2}] = \tilde{O}(N^{1/4})$
- \text{Benstra ECM: } Time $L_p[\frac{5}{4}, \frac{1}{4}]$, Space $O(1)$

Pollard $\rho$-1: Time $\tilde{O}(B \log N)$, Space $O(1)$

- When $p-1$ is "B-smooth" = all prime factors $\leq B$.

Takeaways: General strategy is to turn hard problem (e.g. log, factoring) into easy problem (e.g. all finding, linear algebra).
Quantum Cryptanalysis:

Factoring, possible settings:

- Classical → Classical
- Classical → Quantum
- Quantum → Classical
- Quantum → Quantum

Chall (defender) ↔ Adv (attacker)

Today: 
- Quantum?
- For out (see Zhendry)

Saying we had a large QC, then consequences:

- Factoring, RSA → poly time
- ECDLs → "
- FF DVD

- Block cipher → Attacks improve
- Hashing

G Grover Search
- Double key length n bits → 2n bits

Other cool non-trivial attacks if both adv and chall are QC:
- Even-Mansour, AES-GCM broken in poly time! (Regev et al CRYPTO '16)

Many questions about feasibility of these attacks
- e.g. Google QC talk in Jan 2019
  - Shor Factoring RSA 2048: 250,000,000 physical qubits for ~10 hours
  - Today we have QCs with: 9 qubits

"Just a small matter of engineering"?
Post-Quantum Crypto

- To prepare for PQ world, need new PRE mechanisms
  - Digital Sigs
  - Key exchange
- NIST is looking for proposals now! Against quantum attacks
  - Two popular directions
    - Hash-Based Sigs
    - Lattice-based Crypto
Hash-Based Signatures

Surprising fact: Can build sig schemes from sym-key primitives (SHA-256)

Best QC attacks on SHA-256, AES 256 still ~ 2^{128} cost

So suggests building sigs from hash functions.

PRO: No need for new security assumptions

CON: Large signatures

EC-Schnorr 60 B

RSA-2048 256 B

SPHINCS 41,000 B

If you have a better idea for hash-based sigs, NIST would like to talk to you!

Recall syntax for signature scheme on msg space \( M \)

\[ \text{Gen}(1^k) \rightarrow (pk, sk) \]

\[ \text{Sign}(sk, m) \rightarrow \sigma \]

\[ \text{Verify}(pk, m, \sigma) \rightarrow \{0, 1\} \]

Informally:

- Correctness: \( \forall m \in M, (pk, sk) \text{ from } \text{Gen}, \text{ Verify accepts valid signature.} \)

- Security: Adv can't produce a new valid \((m, \sigma)\) pair, even after seeing many sigs on msgs of her choosing.

---

\[\text{Chal} \quad \begin{array}{c}
\rightarrow pk \\
\leftarrow m \\
\downarrow \\
\text{acc/\neg acc}
\end{array} \quad \begin{array}{c}
\downarrow \\
\sigma = \text{Sign}(sk, m) \\
\leftarrow (m, \sigma)
\end{array} \quad \text{Adv} \]

Adv wins if \((m, \sigma)\) not result of query and is valid.
LaPorte: One-Time Signatures (1979)

- Sign once (in sense, adv. gets one signing query)
- Sign twice \( \rightarrow \) Broken
- Essentially: Give away part of \( \text{sk} \) w/ each sig.
- Only needs a OWF - think: SHA-256

Construction uses \( H: \{0,1\}^* \rightarrow \{0,1\}^n \) (OWF)

Message space: \( m \in \{0,1\}^n \)

Signatures: \( \Sigma \in \{0,1\}^{2n} \)

\[ \text{Gen}(1^n) \rightarrow (pk, sk) \]

\[ \begin{align*}
  sk &= \left( x_{0,1}, x_{0,2}, \ldots, x_{0,n} \right) \\
  pk &= \left( H(x_{0,1}), \ldots, H(x_{0,n}) \right)
\end{align*} \] \( \in \{0,1\}^{2n} \)

\[ \begin{align*}
  \Sigma &= \left( H(x_{1,1}), \ldots, H(x_{1,n}) \right) \\
  \in \{0,1\}^{2n}
\end{align*} \]

\[ \text{Sign}(sk, m) \rightarrow \sigma \]

\[ \text{Write} \quad m = m_1 m_2 m_3 \ldots m_n \in \{0,1\}^n \]

\[ \text{Output} \quad (x_{m,1}, x_{m,2}, \ldots, x_{m,n}) \in \{0,1\}^{2n} \]

\( \Sigma \) Naive implementation: \( 4 \sim 8 \text{KB} \) sig. len

\[ \text{Ver}(pk, m, \sigma) \rightarrow \{0,1\} \]

\[ \text{For } i = 1, \ldots, n: \]

\[ \text{Check } pk_{m,i} = H(\sigma_i) \]

Example to sign: \( m = 00101 \)

\[ \begin{align*}
  sk &= \left( x_{0,1}, x_{0,2}, x_{0,3}, x_{0,4}, x_{0,5} \right) \\
  pk &= \left( H(x_{0,1}), \ldots, H(x_{0,5}) \right)
\end{align*} \]
Lampert: Sign

Security of idea: If I add that Forge's sig, can use it to invert H.

Claim: Two signatures make it insecure.
Adv gets sig on 0^n and 1^n.

Optimizations can shrink sigs to ~λn/2 bits (instead of n).
6 Compare with ~2λ for ECDSA/EC Schnorr

Problem: Want to sign many messages!

Solution 1: Stateful sigs
- Put many Lampert sigs in Merkle tree
- Publish tree root
- Use each leaf pub to sign on msg

Problems: too stateful
* keysgen times large

Solution 2: "Signatures all the way down..."
Fill Hash-Based Signatures

First, see that if we have a OTS for n-bit msgs, we can sign all msgs in \( \{0,1\}^* \) using a CHHF.

\[
\text{Sign}(sk, m) := \text{Sign}_{\text{OTS}}(sk_{\text{OTS}}, H(m)).
\]

\( \{0,1\}^* \)  
H outputs n bits.

If \( n \geq 2\lambda \), this is still a secure sig scheme.

Idea: Use a tree of public keys for OTS scheme.
- The pk is the root of the tree
- A signature includes all sks on path to a random leaf of tree.

- Merkle's original tree... classic!

![Diagram of a Merkle tree](attachment:image.png)
Full Hash-Based Sigs

Sketch of signing alg

1) Pick random leaf in tree. Say pk_{000}.
2) Use pk_{00} to sign msg m.
3) Use parent of pk_{000} to sign pk_{00} and sibling.
4) Use parent of pk_{01} to sign pk_{01} and sibling.
5) Root of tree.

Given
\[
\begin{pmatrix}
 pk_{00} & pk_{01} & g_0 \\
 pk_{010} & pk_{011} & g_{0,10} \\
 m & & g_{0,11}
\end{pmatrix}
\]
can bind sig on m back to root public key.

Signature size is $O(n^3)$ bits; $O(n^2)$ ors of $O(n^2)$ bits each.

Naive construction would require exponential-time setup to build tree.
Actual construction uses PRF to generate all intermediate secret keys.
Full OTS

Key gen routine: Samples a random PRF key $k$ and stores it as part of $sk$. 

$pk$ for tree consists of two OTS pk's.

\[ \begin{align*} & pk_0 \mid pk_1 \\
& \downarrow \quad \downarrow \\
& pk_{00} \mid pk_{01} \quad pk_{01} \mid pk_{11} \\
& \downarrow \quad \downarrow \quad \downarrow \\
& m \end{align*} \]

To sign: Choose random $r_1 \in \{0, \ldots, 2^n\}$ 

- Set $\sigma_{010}$ 
- $(sk_{010}, sk_{011}) \leftarrow \text{PRF}(k, 010)$ 
- $pk_{010} \leftarrow H(sk_{010})$ 
- $pk_{011} \leftarrow H(sk_{011})$ 
- $\sigma_m \leftarrow \text{SignOTS}(sk_{010}, m)$ 
- $(sk_{00}, sk_{01}) \leftarrow \text{PRF}(k, 01)$ 
- $pk_{00} \leftarrow H(sk_{00})$ 
- $pk_{01} \leftarrow H(sk_{01})$ 
- $\sigma_{01} \leftarrow \text{SignOTS}(sk_{01}, pk_{01} || pk_{01})$ 
- $\sigma_0 \leftarrow \text{SignOTS}(sk_0, pk_{00} || pk_{01})$ 
- Output $(\sigma_0, \sigma_{01}, \sigma_m, pk_{01}, pk_{11}, pk_{00}, pk_{01})$
Full Hash-Based Sigs

Why secure?
- Each pk inside tree only used to sign a single msg.
- As long as no collisions in choice of random leaf, each leaf pk only used to sign one msg.

Still problematic... lots of hashes, annoying sig size.
- Some optimizations improve sig size
  - Naive: 2 MB, SPHINCS: ~40 KB
See Boneh-Sharp for many details