June 6: Post-Quantum Crypto

REMINDER: Project!!!

But first, review!

<u>Review</u>: Classical Cryptanalysis

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

Given $g, h = g^x$ find x

Idea: Find collision in $F(\alpha, \beta) = g^\alpha h^\beta \in \mathbb{G}$

BSGS: Time $\tilde{O}(\sqrt{p})$, Space $\tilde{O}(\sqrt{p})$

Pollard Rho: Time $\tilde{O}(\sqrt{p})$, Space $\tilde{O}(1)$

$\Big\}$ Collision finding

polylog(p)

Shoup: Can't do better "generically"

$\hookrightarrow$ $\exists$ faster alg's for specific $\mathbb{G}$ (e.g. GNFS)

Factoring: Split $N = pq$

Trial Division Time $\tilde{O}(\sqrt{N})$, Space $\tilde{O}(1)$

Pollard Rho Time $\tilde{O}(N^{1/4})$, Space $\tilde{O}(1)$

Dixon Time $L_N[\frac{1}{2}, \cdot]$, Space $L_N[\frac{1}{2}, \cdot]$

$\hookrightarrow L_p[\frac{1}{2}] = e^{\sqrt{\log N \log \log N}}$

Lenstra ECM Time $L_p[\frac{1}{3}]$, Space $\tilde{O}(1)$

$\hookrightarrow$ Smallest prime factor!

$\Big\}$ General purpose

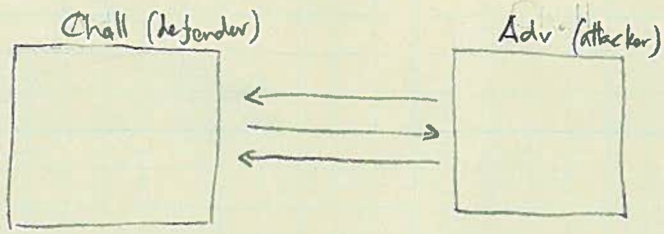
Pollard p-1 Time $\tilde{O}(B \log N)$, Space $\tilde{O}(1)$ $\Big\}$ Special purpose

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

Take away: General strategy is to turn hard problem (e.g. dlog, factoring) into easy problem (e.g. coll finding, lin algebra).

Quantum Cryptanalysis

Four possible settings...



Classical	← bits →	Classical	
Classical	"	Quantum	} Today
Quantum	← qubits →	Quantum	} Far out (see Zhandry)

Say we had a large QC, then consequences

Factoring, RSA → poly time
ECDlog → "
FF Dlog → "
} Shor

Block cipher → Attacks improve
Hashing →
} Grover search

↳ Double key length n bits → 2n bits

Other cool non-trivial attacks if both adv and chal are QC:

Even-Mansour, AES-GCM broken in poly time! (Kaplan et al CRYPTO '16)

Many questions about feasability of these attacks

e.g. Google QC talk in Jan 2017

Shor Factoring RSA 2048 : 250,000,000 physical qubits for ~10 hours

Today we have QCs w/: 9 qubits

"Just a small matter of engineering" ?

Post-Quantum Crypto

- To prepare for PQ world, need new PKE mechanisms
  Digital Sigs
  Key exchange

- NIST is looking for proposals now!

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 sha256, AES 256 still $\sim 2^{128}$ cost

↳ Suggests building sigs from hash functions.

PRO: No need for new funny 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 schemes over msg space $\mathcal{M}$

$$\mathrm{Gen}(1^\lambda) \to (pk, sk)$$

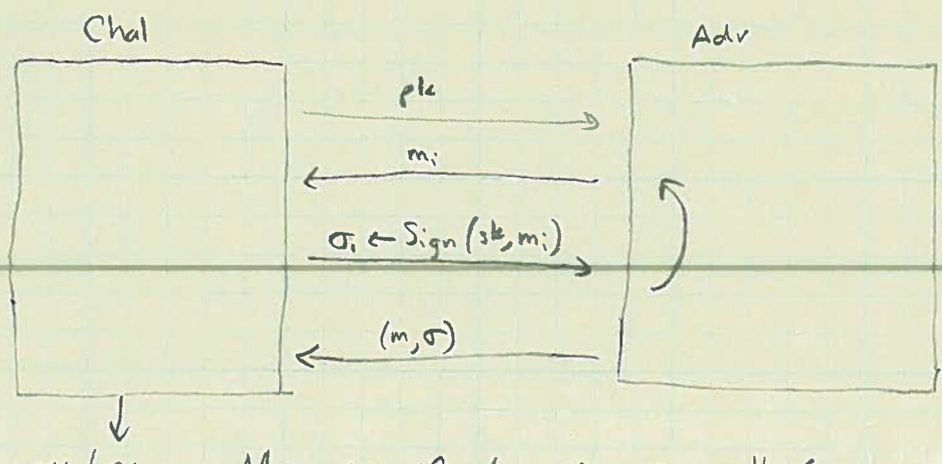
$$\mathrm{Sign}(sk, m) \to \sigma$$

$$\mathrm{Verify}(pk, m, \sigma) \to \{0,1\}$$

Informally

Correctness: $\forall m \in \mathcal{M}$, (pk, sk) from Gen, Verify accepts valid signatures.

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



Adv wins if (m, σ) not result of query and is valid.

Lamport One-Time Signatures (1979)

- Sign once (in sec game, adv gets one signing query)
- Sign twice → Broken
- Essentially: Give away part of sk w/ each sig
- Only needs a OWF - think: SHA-256

Construction uses $H: \{0,1\}^\lambda \to \{0,1\}^\lambda$ (OWF)

Message space: $\mathcal{M} = \{0,1\}^n$

Signatures $\Sigma = \{0,1\}^{n\lambda}$

$\mathrm{Gen}(1^\lambda) \to (pk, sk)$

$$sk = \begin{pmatrix} x_{0,1} & x_{0,2} & x_{0,3}, & \dots, & x_{0,n} \\ x_{1,1} & x_{1,2} & x_{1,3} & \dots, & x_{1,n} \end{pmatrix} \xleftarrow{R} \{0,1\}^{2\lambda n}$$

$$pk = \begin{pmatrix} H(x_{0,1}), & \dots, & H(x_{0,n}) \\ H(x_{1,1}), & \dots, & H(x_{1,n}) \end{pmatrix} \in \{0,1\}^{2\lambda n}$$

$\mathrm{Sign}(sk, m) \to \sigma$

Write $m = m_1 m_2 m_3 \dots m_n \in \{0,1\}^n$

Output $(x_{m_1,1}, x_{m_2,2}, \dots, x_{m_n,n}) \in \{0,1\}^{\lambda n}$

↳ Naive implementation: 4-8kB sig len

$\mathrm{Ver}(pk, m, \sigma) \to \{0,1\}$

For $i = 1, \dots, n$:

Check $pk_{m_i, i} \stackrel{?}{=} H(\sigma_i)$

Example to sign $m = 00101$

$$sk = \begin{pmatrix} x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} & x_{0,5} \\ x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \end{pmatrix}$$

Lamport Sigs

Security pf idea: If $\exists$ adv that forges 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 $\sim \lambda n/2$ bits (instead of λn).

↳ Compare with $\sim 2\lambda$ for ECDSA/EC Schnorr

Problem: Want to sign many messages!

Solution 1: Stateful sigs

- Put many Lamport pks in Merkle tree
- Publish tree root
- Use each leaf pk to sign one msg

Problems: * state :(
* keygen time is large

Solution 2: "Signatures all the way down..."

Full Hash-Based Signatures

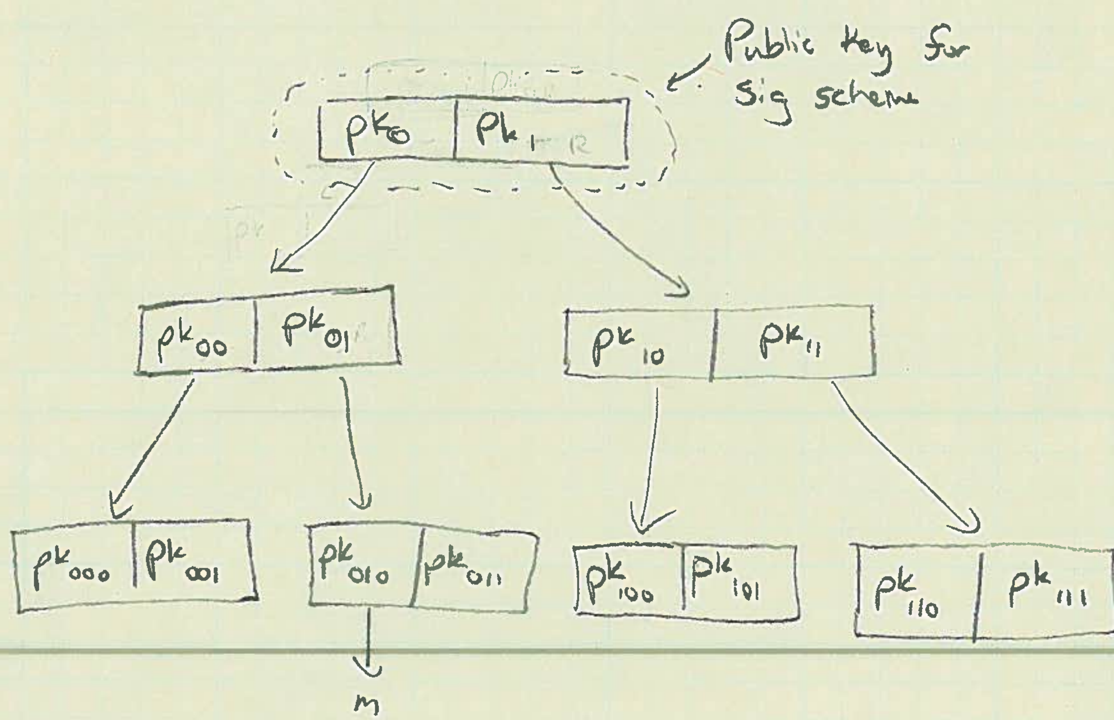
First, see that if we have a OTS for n-bit msgs, we can sign all msgs in $\{0,1\}^*$ using a CRHF:

$$\mathrm{Sign}(sk, \underset{\substack{\uparrow \\ \{0,1\}^*}}{m}) := \mathrm{Sign}_{OTS}(sk_{OTS}, \underbrace{H(m)}_{H \text{ outputs } n \text{ bits.}}).$$

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 pks on path to a random leaf of tree.

Merkle's original tree... classic!



Full Hash-Based Sigs

Sketch of signing alg

1) Pick random leaf in tree. Say pk_{010}.
2) Use pk_{010} to sign msg m.
3) Use parent of pk_{010} to sign pk_{010} and sibling
4) Use parent of pk_{01} to sign pk_{01} and sibling
   ↳ Root of tree.

Given $\begin{pmatrix} pk_{00} & pk_{01} & \sigma_0 \\ pk_{010} & pk_{011} & \sigma_{01} \\ & m & \sigma_m \end{pmatrix}$ Can bind sig on m back to root public key.

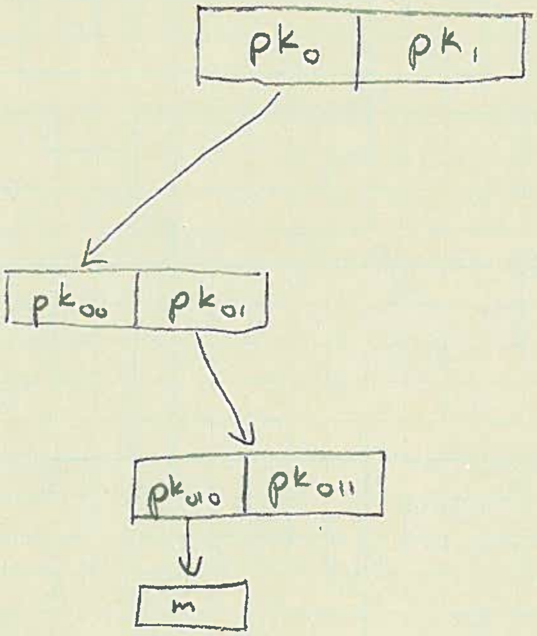
Signature size is $O(\lambda^3)$ bits: $O(\lambda)$ OTS of $O(\lambda^2)$ bits each

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 & stores as part of sk.
* pk for tree consists of two OTS pks.



To sign: Choose random int r_i in $\{0, \dots, 2^{2\lambda}\}$

Say 010

$(sk_{010}, sk_{011}) \leftarrow PRF(k, 010)$

$pk_{010} \leftarrow H(sk_{010})$

$pk_{011} \leftarrow H(sk_{011})$

$\sigma_m \leftarrow Sign_{OTS}(sk_{010}, m)$

$(sk_{00}, sk_{01}) \leftarrow PRF(k, 01)$

$pk_{00} \leftarrow H(sk_{00})$

$pk_{01} \leftarrow H(sk_{01})$

$\sigma_{01} \leftarrow Sign_{OTS}(sk_{01}, pk_{010} \| pk_{01})$

$\sigma_0 \leftarrow Sign_{OTS}(sk_0, pk_{00} \| pk_{01})$

Output $(\sigma_0, \sigma_{01}, \sigma_m, pk_{01}, pk_{11}, pk_{010}, pk_{011})$

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-Shoup for many details