Today’s Agenda

- OWF vs. OWF families
- Candidate OWFs and families
  - Factoring-based: Multiplication, RSA, Squaring
  - Lattice-based
  - Ad-hoc

1 One-way Functions Families

Recall that \( f : \{0,1\}^k \rightarrow \{0,1\}^{\text{poly}(k)} \) is a OWF if \( f \) is:

1. Efficient to compute
2. Hard to invert

We have seen that the key generation algorithm for a public-key cryptosystem should be a OWF. Given the \( PK \), we want it to be hard to find the corresponding randomness that created it so as to keep the secret key secret. Same should hold for Encrypt(\( PK, \cdot \)) because we want it to be hard to retrieve the message that is encrypted.

Instead of considering single one-way functions, we will now look at what are one-way function families.

Definition 1 (OWF Family). A triple of algorithms \((\text{KeyGen}, \text{Domain-Sampling}, \text{Eval})\) constitute a OWF family if:

- \( \text{KeyGen}(1^k) \) outputs an evaluation key \( K \)
- \( \text{Domain-Sampling}(K) \) is a probabilistic algorithm that outputs \( x \)
- \( \text{Eval}(K, x) \) outputs \( f_K(x) \)

We also need that all the three algorithms are poly-time and we define one-wayness as follows:

\[
\Pr[K \leftarrow \text{KeyGen}(1^k) ; x \leftarrow \text{Domain-Sampling}(K) ; y = f_K(x) = \text{Eval}(K, x) ; x' \leftarrow A(1^k, K, y) : f_K(x') = y] = \gamma(k)
\]

Note that the OWF definition we have seen so far is a special case of the definition above. So we can rewrite our definition for OWF as follows: \( f \) is a OWF if \((\text{Identity}, \{0,1\}^k, f)\) is a OWF family.
2 Candidates for OWF

- **Candidate 1**: On input \( x \) such that \( x = x_1(\circ)x_2 \), output \( y = x_1.x_2 \) which is multiplication of integers \( x_1, x_2 \).

An obvious way to invert this would be, given \( y \), we always have \( f(000 \ldots 1(\circ)y) = y \) since any integer multiplied by 1 will give us the same integer and so we can output \( x_1 = 00 \ldots 1 \) and \( x_2 = y \). The problem with this is that if \( f \) is length preserving or if \( f \) maps to anything less than \( 2k \) bits, we cannot invert \( f \) this way.

What about if \( y \) is a square? You can invert it as \( x_1 = x_2 = \sqrt{y} \). So now the question is how likely is it that a random \( y \) is a square and given a square how easy is it to find the square-root? Asymptotically, the density of squares occurring decreases exponentially, so the probability of sampling a square goes down exponentially.

**CONCLUSION**: This candidate as it is is not a OWF because if \( x_1, x_2 \) happen to have small factors then it is easy to factor \( y \)

- **Candidate 2**: Our input is \( x = x_1 \circ x_2 \circ \ldots \circ x_{l2} \circ x'_{1} \circ x'_{2} \circ \ldots \circ x'_{l2} \) such that \( 2l^2.l = k \) i.e. \( l = \sqrt[3]{\frac{k}{2}} \) and each \( x_i, x'_i \) is an \( l \)-bit string. Essentially, we are parsing our input \( x \) as a concatenation of \( 2l^2 \) integers. Now we find first \( i \) such that \( x_i \) is a prime and first \( j \) such that \( x'_j \) is a prime. Output \( y = x_i.x'_j \) if \( i,j \) exist and \( 1^k \) otherwise.

**Theorem 1** (Prime-number Theorem (Informal)). *One in every \( l \) \( l \)-bit integers is prime asymptotically.*

\[ \exists c \text{ s.t. } \lim_{l \to \infty} \frac{\text{Number of primes} < 2^l}{2^l} \geq \frac{1}{cl} \]

So the probability that you fail in outputting a \( y \) which is product of two primes is the probability that you do not find a prime in \( l^2 \) \( x_i \)'s and also not find a prime in the \( x'_j \)'s which is negligible. For more about distribution of primes, refer to Chapter 5 of Victor Shoup’s *A Computational Introduction to Number Theory and Algebra*

The OWF candidate described here is the RSA *modulus generation* algorithm which is also used in GM cryptosystem. This is heavily used in practice and its recommended modulus length in practice is 2048 where \( l = 1024 \) according to previous description. The best known attack for this will take \( O(2^{100}) \) time as well as huge amounts of space. This is believed to be infeasible given the current state-of-art in factoring algorithms.

- **Candidate 3**: RSA Permutation as OWF Family

  1. Key-generation: On input \( 1^k \),
     - Find \( k \)-bit primes \( p \) and \( q \), let \( N = pq \).
     - Find \( e, d \) s.t. \( ed \equiv 1 \mod (p - 1)(q - 1) \) for \( 1 \leq e \leq N, 1 \leq d \leq N \).
     - Output \( K = (N, e) \)
  2. Domain-sample\((N,e)\): Output \( x \leftarrow \mathbb{Z}_N^* \) which means sample \( x \) from the numbers that are co-prime to \( N \) from \( \{1, \ldots, N - 1\} \)
  3. Eval\((N,e),x\): Output \( x^e \mod N \)
  4. Invert\((N,d),y\): Output \( y^d \mod N \)

  We will postpone the proof of correctness.

Lec 6: OWF Families and Candidate OWF-2
Given $x \in \mathbb{Z}_N^*$, $1 \leq e \leq N$, how do you efficiently compute $x^e \mod N$? Exponentiate using repeated squaring:

$$\prod_{i \in [l]} x^{2^i} = x^{\sum 2^i} = x^e \mod N$$

For the RSA permutations, our implicit assumption is that factoring is hard (NP-hard). It is possible that even if RSA broken, factoring is still hard. Also, there is no formal reduction from RSA to factoring but all the known attacks on RSA, factor the modulus $N$ first and do not exploit the fact that something simpler than factoring may work.

- **Candidate 4: Blum Permutation**
  
  1. Key generation: RSA modulus $N = pq$ where $p \equiv q \equiv 3 \mod 4$
  2. Domain-sample: Output $x = z^2 \mod N$ for $z \leftarrow \mathbb{Z}_n^*$
  3. Eval($N, x$): Output $x^2 \mod N$
  4. Invert($(p, q), y$): Output $\sqrt{y}$

- **Candidate 5: By Ajtai’96**
  
  Key $K$ is a matrix $A \in \mathbb{Z}_q^{n \times m}$. For an $m$-bit $x$, define

  $$f_A(x) = Ax \mod q$$

  Here $q = poly(n) = poly(m)$ and $m \geq n \log q$

  This is an excellent OWF but the reason it is not used much in practice is that is is impractical to have a huge matrix like this as a key. The proof that this is a OWF uses elements from lattices and lattice-based cryptography. The nice thing about lattice-based cryptography is that it is secure even if efficient quantum computers exist which is not necessarily true for rest of the cryptography.

- **Ad-hoc OWF:** Another way of designing heuristically good OWF could be that you design a OWF, make it public and have competitions for people to break it. If the OWF is unbroken after a reasonable amount of time then you can say with some confidence that what you designed is a reasonably good OWF. NIST (National Institute of Standards and Technology) often conducts such competitions to come up with good OWF in the form of hash functions. The popular hash function SHA-3 was a result of one such competition by NIST.