Today:

– Introduction to the class.
– Examples of concrete physical attacks on RSA
– A computational approach to cryptography
– Pseudorandomness

1 What are Physical Attacks

– Tampering/Leakage attacks
– Issue of how we model adversaries in cryptography

2 Physical Attacks on RSA

This section is based on the survey of Boneh [1].

We begin by describing a simplified version of RSA encryption. Let $N = pq$ be the product of two large primes of the same size ($n/2$ bits each). A typical size for N is $n = 1024$ bits, i.e. 309 decimal digits. Each of the factors is 512 bits. Let e, d be two integers satisfying $ed = 1 mod \phi(N)$, where $\phi(N) = (p-1)(q-1)$ is the order of the multiplicitave group $\mathbb{Z}_N^*$. We call N the *RSA modulus*, e the *encryption exponent*, and d the *decryption exponent*. The pair $\langle N, e\rangle$ is the *public key*. As its name suggests, it is public and is used to encrypt messages. The pair $\langle N, d\rangle$ is called the *secret key* or *private key* and is known only to the recipient of encrypted messages. The secret key enables decryption of ciphertexts.

A *message* is an integer $M \in \mathbb{Z}_N^*$. To encrypt M, one computes $C = M^e \mod N$. To decrypt the ciphertext, the legitimate receiver computes $C^d \mod N$. Indeed $C^d = M^{ed} = M(\mod N)$, where the last equality follows by Euler's theorem.

–RSA function vs. cryptosystem –"Break RSA" means doing better than brute-force search.

2.1 Timing Attack on RSA

Attack of Kocher [3].

– Smartcard that stores a private RSA key. Since the card is tamper resistant, an attacker may not be able to examine its contents and expose the kye. However, a clever attack shows that by precisely measuring the time it takes the smartcard to perform an RSA decryption (or signature), can quickly discover the private decryption exponent d.
– Assume we have an RSA implementation based on the "repeated squaring algorithm." Let $d = d_n d_{n-1} \dots d_0$ be the binary representation of d:

$$d = \sum_{i=0}^{n} 2^i d_i \text{ with } d_i \in \{0, 1\}.$$

The *repeated squaring algorithm* computes $C = M^d \mod N$, using at most $2n$ modular multiplications. Based on observation that $C = \Pi_{i=0}^{n} M^{2^i d_i} \mod N$. The algorithm works as follows:

1. Set $z = M, C = 1$. For $i = 0, \dots, n$, do the following:
2. If $d_i = 1$ set $C = C \cdot z \mod N$
3. Set $z = z^2 mod N$.
4. At the end, C has the value $M^d \mod N$.

The Attack:

1. As the smartcard to generate signatures on a large number of random messages $M_1, \ldots, M_k \in \mathbb{Z}_N^*$ and measure the time T_i it takes to generate each signature.
2. Recover the bits of d one at a time beginning with the least significant bit:
   - d is odd, so $d_0 = 1$.
   - Finding d_1: Initially, $z = M^2 modN$ and $C = M$. If $d_1 = 1$, the smartcard computes the product $C \cdot z = M \cdot M_i^2 \mod N$. Otherwise it does not.
   - Let t_i be the time it takes the smartcard to compute $M_i \cdot M_i^2 \mod N$. The t_i's differ from each other since the time to compute $M_i \cdot M_i^2 \mod N$ depends on the value of M_i. Measure the t_i's offline (prior to mounting attack) once he obtains the physical specifications of the card.
   - When $d_1 = 1$, the two ensembles $\{t_i\}$ and $\{T_i\}$ are correlated. For instance, if for some i, t_i is much larger than its expectation, then T_i is also likely to be larger than its expectation.
   - On the other hand, if $d_1 = 0$, the two ensembles $\{t_i\}$ and $\{T_i\}$ behave as independent random variables.
   - By measuring the correlation, Marvin can determine whether d_1 is 0 or 1. Continuing in this way, can recover $d_2, d_3, \ldots$.

2.2 Defense to Timing Attack

Approach due to Rivest, based on *blinding*.

- Prior to decryption of M the smartcard picks a random $r \in \mathbb{Z}_N^*$
- Computes $M' = M \cdot r^e \mod N$.
- Apply d to M' and obtain $C' = (M')^d \mod N$.
- Set $C = C'/r \mod N$.

With this approach, the smartcard is applying d to a random message M' unknown to attacker. As a result, cannot mount the attack.

2.3 Tampering Attack on RSA

Attack of Boneh, Demillo, Lipton [2].

- Implementations of RSA decryption and signatures frequently use the Chinese Remainder Theorem to speed up the computation of $M^d \mod N$.
- Instead of working modulo N, the signer first computes the signature modulo p and q and then combines the results using the Chinese Remainder Theorem.

CRT

1. Compute:
$$C_p = M^{d_p} \mod p$$
   and
$$C_q = M^{d_q} \mod q$$
   where $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$.
2. Obtain the signature C by setting
$$C = T_1 C_p + T_2 C_q (\mod N),$$
   where $T_1 = 1 \mod p, 0 \mod q$ and $T_2 = 0 \mod p, 1 \mod q$.

4 times speedup

The Attack:

1. Suppose that while generating a signature, a glitch on Bob's computer causes it to miscalculate in a single instruction.
2. Given an invalid signature, an adversary can easily factor Bob's modulus N.
3. (Version of attack described by A.K. Lenstra) Suppose a single error occurs, exactly one of C_p or C_q will be computed incorrectly. Say C_p is correct, but $\hat{C}_q$ is not.
4. The resulting signature is $\hat{C} = T_1 C_p + T_2 \hat{C}_q$.
5. Once the adversary receives $\hat{C}$, he knows it is a false signature since $\hat{C}^e \neq M \mod N$. However, notice that

$$\hat{C}^e = M \mod p$$

while

$$\hat{C}^e \neq M \mod p$$

6. As a result, $gcd(N, \hat{C}^e - M)$ exposes a nontrivial factor or N.

2.4 Defense to Tampering Attack

- Random padding to signing defeats the attack.
- Simpler defense is for Bob to check the generated signature.
- Many systems, including a non-CRT implementation of RSA can be attacked using random faults. However, these results are far more theoretical.

3 A Computational Approach to Cryptography

Not necessary to use a perfectly-secret encryption scheme, but it instead suffices to use a scheme that cannot be broken in "reasonable time" with any "reasonable probability of success."

A cryptographic scheme does not need to guarantee perfect security (info-theoretically) but just to be "practically" unbreakable.

How to formalize the fact that a cryptographic scheme is "practically" unbreakable?

Two relaxations of Computational Approach over perfect security:

1. Security is only preserved against *efficient* adversaries that run in a feasible amount of time.
2. Adversaries can potentially succeed with some *very small probability* (that is small enough so that we are not concerned that it will ever really happen).

Two approaches to defining the above: Concrete approach and Asymptotic approach.

The Concrete Approach: quantifies the security of a given cryptographic scheme by explicitly bounding the maximum success probability of any adversary running for at most some specified amount of time. Let t, ε be positive constants with $\varepsilon \leq 1$. Then a concrete definition of security would take the following form:
"A scheme is (t, ε)-secure if every adversary running for time at most t succeeds in breaking the scheme with probability at most ε.

EXAMPLE: Modern private-key encryption schemes are generally assumed to give almost optimal security in the following sense: when the key has length n, an adversary running in time t (measured in, say, computer cycles) can succeed in breaking the scheme with probability at most $t/2^n$. Computation on the order of $t = 2^{60}$ is barely within reach today. Running on a $1GHz$ computer (that executes 10^9 cycles per second), 2^{60} CPU cycles require $2^{60}/10^9$ seconds, or about 35 years. Using many supercomputers in parallel may bring this down to a few years. A typical value for the key length, might be $n = 128$. The difference between 2^{60} and 2^{128} is a multiplicative factor of 2^{68} which is a number containing about 21 decimal digits. To get a feeling for how big this is, note that according to physicists' estimates the number of seconds since the big bang is in the order of 2^{58}.

The Asymptotic Approach:

- rooted in complexity-theory, running time of the adversary as well as its success probability are *functions* of some parameter, not concrete numbers.
- a crypto scheme will incorporate a *security parameter* which is an integer n. When honest parties initialize the scheme, they choose n. The running times are viewed as functions of n.

feasible strategies/efficient algorithms are probabilistic algorithms running in time polynomial in n. Honest parties run in polynomial time and achieve security against polynomial-time adversaries.

negligible success success probabilities *smaller than any inverse polynomial in* n.

"A scheme is secure if every ppt adversary succeeds in breaking the scheme with only negligible probability."

EXAMPLE: Say we have a scheme that is secure. Then it may be the case that an adversary running for n^3 minutes can succeed in "breaking the scheme" with probability $2^{40} \cdot 2^{-n}$ (which is a negligible function of n). When $n \leq 40$ this means that an adversary running for 40^3 minutes (about 6 weeks) can break the scheme with probability 1, so such values are not going to be very useful. Even for $n = 50$ an adversary running for 50^3 minutes (about 3 months) can break the scheme with probability roughly $1/1000$, which may not be acceptable. On the other hand, when $n = 500$ an adversary running for more than 200 years breaks the scheme only with probability roughly 2^{-500}.

Efficient Algorithms and Negligible Success

Efficient Computation computation that can be carried out in probabilistic polynomial time (PPT). An algorithm A is said to run in *polynomial time* if there exists a polynomial $p(\cdot)$ such that, for every input $x \in \{0,1\}^*$, the computation of $\mathsf{A}(x)$ terminates within at most $p(|x|)$ steps.
A *probabilistic* algorithm is one that has the capability of "tossing coins."
Advantage of working with polynomial time: Class of algorithms is closed under composition.
Why *probabilistic polynomial time*?

Generating Randomness

Negligible Success Probability

Proofs by Reduction.

Difficulty of proving unconditional security. A cryptographic scheme that is computationally secure can always be broken given enough time. Unconditional proof of security would require proving a lower bound on the time needed to break the scheme. Currently, unable to prove lower bounds of this type. Would be a breakthrough result in complexity theory.

Paradigm. Assume some *low-level* problem is hard to solve, and then *prove* that the construction in question is secure under this assumption.

Details on Paradigm: Present an explicit *reduction* showing how to convert any efficient adversary $\mathcal{A}$ that succeeds in "breaking" the construction with non-negligible probability into an efficient algorithm $\mathcal{A}'$ that succeeds in solving the problem that was assumed to be hard.

To prove some cyrptographic construction Π is secure:

1. Fix some efficient adversary $\mathcal{A}$ attacking Π. Denote this adversary's success probability by $\varepsilon(n)$.

2. Construct an efficient algorithm $\mathcal{A}'$, called the "reductin" that attempts to solve problem X using adversary $\mathcal{A}$ as a sub-routine. An importnat point here is that $\mathcal{A}'$ knows nothing about "how" $\mathcal{A}$ works; the only thing $\mathcal{A}'$ knows is that $\mathcal{A}$ is expecting to attack Π. So given some input instance x of problem X, our algorithm $\mathcal{A}'$ will simulate for $\mathcal{A}$ an instance of Π such that:
   (a) As far as $\mathcal{A}$ can tell, it is interacting with Π. More formally, the view of $\mathcal{A}$ when it is run as a sub-routine by $\mathcal{A}'$ should be distributed identically to (or at least close to) the view of $\mathcal{A}$ when it interacts with Π itself.
   (b) If $\mathcal{A}$ succeeds in "breaking" the instance of Π that is being simulated by $\mathcal{A}'$, this should allow $\mathcal{A}'$ to solve the instance x it was given, at least with inverse polynomial probability $1/p(n)$.
3. Taken together, $2(a), 2(b)$ imply that if $\varepsilon(n)$ is not negligible, then $\mathcal{A}'$ solves problem X with non-negligible probability $\varepsilon(n)/p(n)$. Sinc $\mathcal{A}'$ is efficient, and runs the ppt adversary $\mathcal{A}$ as a sub-routine, this implies an efficient algorithm solving X with non-negligible probability, contradicting the initial assumption.
4. We conclude that, given the assumption regarding X, no efficient adversary $\mathcal{A}$ can succeed in breaking Π with probability that is not negligible. Stated differently, Π is computationally secure.

4 Pseudorandomness

This notion plays a fundamental role in cryptography. Loosely speaking, a pseudorandom string is a string that looks like a uniformly distributed string, as long as the entity that is "looking" runs in polynomial time. pseudorandomness is a computational relaxation of true randomness.

Important Point: no fixed string can be said to be "pseudorandom". Rather, pseudorandomness actually refers to a distribution on strings, and when we say that a distribution $\mathcal{D}$ over strings of length ℓ is pseudorandom this means that $\mathcal{D}$ is indistinguishable from the uniform distribution over strings of length ℓ.

4.1 Pseudorandom Generators

A pseudorandom generator is a deterministic algorithm that receives a short truly random seed and stretches it into a long string that is pseudorandom. n is the length of the seed that is input and $\ell(n)$ is the output length.

Definition 1. *Let $\ell(\cdot)$ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}^n$, algorithm G outputs a string of length $\ell(n)$. We say that G is a pseudorandom generator if the following two conditions hold:*

1. *(Expansion:) For every n it holds that $\ell(n) > n$.*
2. *(Pseudorandomness:) For all ppt distinguishers D, there exists a negligible function* neg *such that*

$$|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| \leq \mathrm{neg}(n)$$

   where r is chosen uniformly at random from $\{0,1\}^{\ell(n)}$, the seed s is chosen uniformly at random from $\{0,1\}^n$, and the probabilities are taken over the random coins used by D and the choice of r and s.

The function $\ell(\cdot)$ is called the expansion factor of G.

References

1. Dan Boneh. Twenty years of attacks on the rsa cryptosystem. *NOTICES OF THE AMS*, 46:203–213, 1999.
2. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.
3. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.