Lecture 22: RSA Encryption

Recall: RSA Assumption

- We pick two primes uniformly and independently at random $p, q \xleftarrow{\$} P_n$
- We define $N = p \cdot q$
- We shall work over the group $(\mathbb{Z}_N^*, \times)$, where $\mathbb{Z}_N^*$ is the set of all natural numbers $< N$ that are relatively prime to N, and $\times$ is integer multiplication $\mod N$
- We pick $y \xleftarrow{\$} \mathbb{Z}_N^*$
- Let $\varphi(N)$ represent the size of the set $\mathbb{Z}_N^*$, which is $(p-1)(q-1)$
- We pick any $e \in \mathbb{Z}_{\varphi(N)}^*$, that is, e is a natural number $< \varphi(N)$ and is relatively prime to $\varphi(N)$
- We give (n, N, e, y) to the adversary $\mathcal{A}$ as ask her to find the e-th root of y, i.e., find x such that $x^e = y$

RSA Assumption. For any computationally bounded adversary, the above-mentioned problem is hard to solve

Recall: Properties

- The function $x^e \colon \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is a bijection for all e such that $\gcd(e, \varphi(N)) = 1$
- Given (n, N, e, y), where $y \xleftarrow{\$} \mathbb{Z}_N^*$, it is difficult for any computationally bounded adversary to compute the e-th root of y, i.e., the element $y^{1/e}$
- But given d such that $e \cdot d = 1 \mod \varphi(N)$, it is easy to compute $y^{1/e}$, because $y^d = y^{1/e}$

Now, think how we can design a key-agreement scheme using these properties. Once the key-agreement protocol is ready, we can use a one-time pad to create an public-key encryption scheme.

Key-Agreement

First, Alice and Bob establish a key that is hidden from the adversary

Alice		Bob
		$p, q \xleftarrow{\$} P_n$
		$N = p \cdot q$
$r \xleftarrow{\$} \mathbb{Z}_N^*$	$\xleftarrow{\mathrm{pk} = (n, N, e)}$	Pick any $e \in \mathbb{Z}_{\varphi(N)}^*$
$y = r^e$	$\xrightarrow{y}$	$\widetilde{r} = y^d$

Note that $r = \widetilde{r}$ and is hidden from an adversary based on the RSA assumption

Public-key Encryption after the Key-Agreement Protocol

Using this key, Alice sends the encryption of $m \in \mathbb{Z}_N^*$ using the one-time pad encryption scheme.

Alice		Bob
$c = m \cdot r$	$\xrightarrow{\quad c \quad}$	$\widetilde{m} = c \cdot \mathrm{inv}(\widetilde{r})$

Since, we always have $r = \widetilde{r}$, this encryption scheme always decrypts correctly. Note that $\mathrm{inv}(\widetilde{r})$ can be computed only by knowing $\varphi(N)$.

Putting the two together: RSA Encryption (First Attempt) I

Alice		Bob
		$p, q \xleftarrow{\$} P_n$
		$N = p \cdot q$
$r \xleftarrow{\$} \mathbb{Z}_N^*$	$\xleftarrow{\text{pk} = (n, N, e)}$	Pick any $e \in \mathbb{Z}_{\varphi(N)}^*$
$y = r^e$		
$c = m \cdot r$	$\xrightarrow{(y, c)}$	$\widetilde{r} = y^d$
		$\widetilde{m} = c \cdot \text{inv}(\widetilde{r})$

Putting the two together: RSA Encryption (First Attempt) II

We emphasize that this encryption scheme work only for $m \in \mathbb{Z}_N^*$. In particular, this works for all messages m that have a binary representation of length less than n-bits, becuase p and q are n-bit primes.

HOWEVER, THIS SCHEME IS INSECURE

Insecurity of the First Attempt

- Let us start with a simpler problem.

  > Suppose I pick an integer x and give $y = x^3$ to you. Can you efficiently find the x?

- Running for for loop with $i \in \{0, \ldots, y\}$ and testing whether $i^3 = y$ or not is an inefficient solution
- However, binary search on the domain $\{0, \ldots, y\}$ is an efficient algorithm
- Then why does the RSA assumption that says "computing the e-th root is difficult if $\varphi(N)$ is unknown" hold? Answer: Because we are working over $\mathbb{Z}_N^*$ and not $\mathbb{Z}$! "Wrapping around" due to the modulus operation while cubing kills the binary search approach.
- However, if x is such that $x^e < N$ then the modulus operation does not take effect. So, if $x < N^{1/e}$ then we can find the e-th root of y!

Insecurity of the First Attempt II

- Now, let us try to attack the "first attempt" algorithm
- Recall that we have $c = m \cdot r$ and $y = r^e$. So, we have $c^e = m^e \cdot r^e$. Now, note that $c^e \cdot \mathrm{inv}(y) = m^e \cdot r^e \cdot y^{-1} = m^e$.
- So, the adversary can compute $c^e \cdot \mathrm{inv}(y)$ to obtain m^e. If $m < N^{1/e}$, then the adversary can use binary search to recover m.
- There is another problem! If Alice is encrypting and sending multiple messages $\{m_1, m_2, \dots\}$, then the eavesdropper can recover $\{m_1^e, m_2^e, \dots\}$. So, she can find which of these $\{m_1^e, m_2^e, \dots\}$ are identical. In turn, she can find out the messages in $\{m_1, m_2, \dots\}$ that are identical (because $x^e \colon \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is a bijection).
- How do we fix these attacks?

RSA Encryption

- Our idea is to pad the message m with some randomness s. The new message $s\|m$, with high probability, satisfies $(s\|m)^e > N$ (that is, it wraps around)
- How does it satisfy the second attack mentioned above (Think: Birthday bound)
- Let us write down the new encryption scheme for $m \in \{0,1\}^{n/2}$

$\mathsf{Enc}_{n,N,e}(m)$:

1. Pick $r \xleftarrow{\$} \mathbb{Z}_N^*$
2. Pick $s \xleftarrow{\$} \{0,1\}^{n/2}$
3. Compute $y = r^e$, and $c = (s\|m) \cdot r$
4. Return (y, c)

Final Optimized RSA Encryption

- Note that masking with r is not helping at all! Let us call $s\|m$ as the payload. An adversary can obtain the "e-th power of the payload" by computing $c^e \cdot y^{-1}$
- So, we can use the following optimized encryption algorithm instead

$\mathsf{Enc}_{n,N,e}(m)$:

1. Pick $s \xleftarrow{\$} \{0,1\}^{n/2}$
2. Return $c = (s\|m)^e$

Looking Ahead: Implementing RSA

Let us summarize all the algorithms that we need to implement RSA algorithm

1. Generating n-bit primes to sample p and q
2. Generating e such that e is relatively prime to $\varphi(N)$, where $N = pq$
3. Finding the trapdoor d such that $e \cdot d = 1 \mod \varphi(N)$