Lecture 24a: Digital Signatures using RSA Assumption

- Bob wants to receive encrypted messages. So, Bob fixes n, the number of bits in the primes he wants to choose. Bob picks two random n-bit primes p and q. Bob computes $N = p \cdot q$. Bob samples a random $e \in \mathbb{Z}^*_{\varphi(N)}$. Bob computes $d \in \mathbb{Z}^*_{\varphi(N)}$ such that $e \cdot d = 1 \mod \varphi(N)$ using the extended GCD algorithm. Bob set $\mathrm{pk} = (n, N, e)$ and $\mathrm{trap} = d$.
- The public-key for Bob pk is broadcast to everyone
- To encrypt a message $m \in \{0,1\}^{n/2}$, Alice runs the $\mathrm{Enc}_p k(m)$ algorithm defined as follows. Alice samples $r \in \{0,1\}^{n/2}$ and computes $c = (r \| m)^e \mod N$. The cipher text is c.
- After receiving a cipher-text $\widetilde{c}$, Bob runs the decryption algorithm $\mathrm{Dec}_{pk,\mathrm{trap}}(\widetilde{c})$. Bob computes $(\widetilde{r}, \widetilde{m}) = \widetilde{c}^d \mod N$.

- **Correctness.** We have seen that this public-key encryption is always correct (relies on the fact that $\gcd(e, \varphi(N)) = 1$)
- **Security.** We have seen that this public-key encryption scheme is secure as long as the randomness r used in every encryption algorithm is distinct against computationally bounded eavesdroppers (relies on the birthday bound and the RSA assumption)

Abstraction

- Recall that we have seen that the function $f_e\colon \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $f_e(x) = x^e \mod N$ is a bijection that is efficient to evaluate. We shall abstract this concept as "Evaluation is efficient"
- Recall that the inverse function $f_e^{-1}\colon \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is efficient to evaluate given d, where $e \cdot d = 1 \mod \varphi(N)$; otherwise, not. We shall abstract this concept as "Inversion is inefficient"
- In a public-key encryption we want that the "encryption algorithm is efficient" and "decryption algorithm is inefficient." So, we used the evaluation of f_e for encryption and the inversion of f_e for decryption.

Digital Signature

- In a digital signature scheme, the signer publishes a public key pk and keeps a trapdoor trap with herself
- Later, if the signer wants to endorse a message m, then she uses an algorithm $\mathsf{Sign}_{\mathsf{pk},\mathsf{trap}}(m)$ to generate a signature σ
- Everyone should be able to verify that "the publisher of the public-key pk endorses the message $\widetilde{m}$ using the signature $\widetilde{\sigma}$" by running the verification algorithm $\mathsf{Ver}_{\mathsf{pk}}(\widetilde{m}, \widetilde{\sigma})$"
- An adversary who sees the public-key pk and a few message-signature pairs $(m_1, \sigma_1), (m_2, \sigma_2), \ldots, (m_k, \sigma_k)$ cannot forge a valid signature σ' on a new message m'

Digital Signatures using RSA I

- First observe that we want "verification to be efficient" and "signing to be inefficient"
- So, using the ideas in the "abstraction slide," the idea is to use "evaluation of f_e" for verification and "inversion of f_e" for signing

Digital Signatures using RSA II

- Alice decides to endorse messages using n-bit primes. Alice picks two random n-bit prime numbers p, q. Alice computes $N = p \cdot q$ and samples a random $e \in \mathbb{Z}^*_{\varphi(N)}$. Alice computes d such that $e \cdot d = 1 \mod \varphi(N)$. Alice sets $\mathrm{pk} = (n, N, e)$ and $\mathrm{trap} = d$
- To sign a message $m \in \{0,1\}^n$, Alice runs $\mathrm{Sign}_{\mathrm{pk},\mathrm{trap}}(m)$ defined as follows. Compute $\sigma = m^d \mod N$.
- To verify a message-signature pair $(\widetilde{m}, \widetilde{\sigma})$, Bob runs the verification algorithm $\mathrm{Ver}_{\mathrm{pub}}(\widetilde{m}, \widetilde{\sigma})$ defined as follows. Output $\widetilde{m} == \widetilde{\sigma}^e \mod N$.

THIS SCHEME IS INSECURE!

Attack on the Previous Scheme

- Pick any $\sigma' \in \mathbb{Z}_N^*$
- Compute $m' = (\sigma')^e \mod N$
- Note that this is an efficient attack
- Note that we did not even need to see any other message-signature pairs
- Although, we do not have any "control" over the message. It is a valid forgery, nonetheless

Fixing our original construction

- We want to use the fact that in the previous forgery attack, the adversary did not have any control over the message that was being signed
- So, here is the idea underlying the fix. We shall pick a random $r \in \{0,1\}^{n/2}$ and include r in the public-key pk. To sign a message $m \in \{0,1\}^{n/2}$, we compute $(r\|m)$ and compute the signature $\sigma = (r\|m)^d \mod N$. To verify a message-signature pair $(\widetilde{m}, \widetilde{\sigma})$, Bob (the verifier) checks $(r, \widetilde{m}) == (\widetilde{\sigma})^e \mod N$
- The formal scheme is presented next

Fixing our original construction II

Gen(1^n):

- Pick random n-bit primes p and q.
- Compute N and $\varphi(N)$
- Sample $e \in \mathbb{Z}^*_{\varphi(N)}$
- Compute d such that $e \cdot d = 1 \mod \varphi(N)$
- Sample random $r \in \{0,1\}^{n/2}$
- Return $\mathsf{pk} = (n, N, e, r)$ and $\mathsf{trap} = d$

$\text{Sign}_{\text{pk,trap}}(m)$:

- Return $(r \| m)^d \mod N$

$\text{Ver}_{\text{pk}}(\widetilde{m}, \widetilde{\sigma})$:

- Return $(r \| \widetilde{m}) == \widetilde{\sigma}^e \mod N$

In the next lecture, we shall learn how to sign arbitrary-length messages $m \in \{0, 1\}^*$