Introduction to Cryptography, Part II

Alice and Bob

- Alice wants to communicate securely with Bob
- (Cryptographers frequently speak of Alice and Bob instead of A and B. . .
- What key should she use?

Pre-Arranged Key Lists?

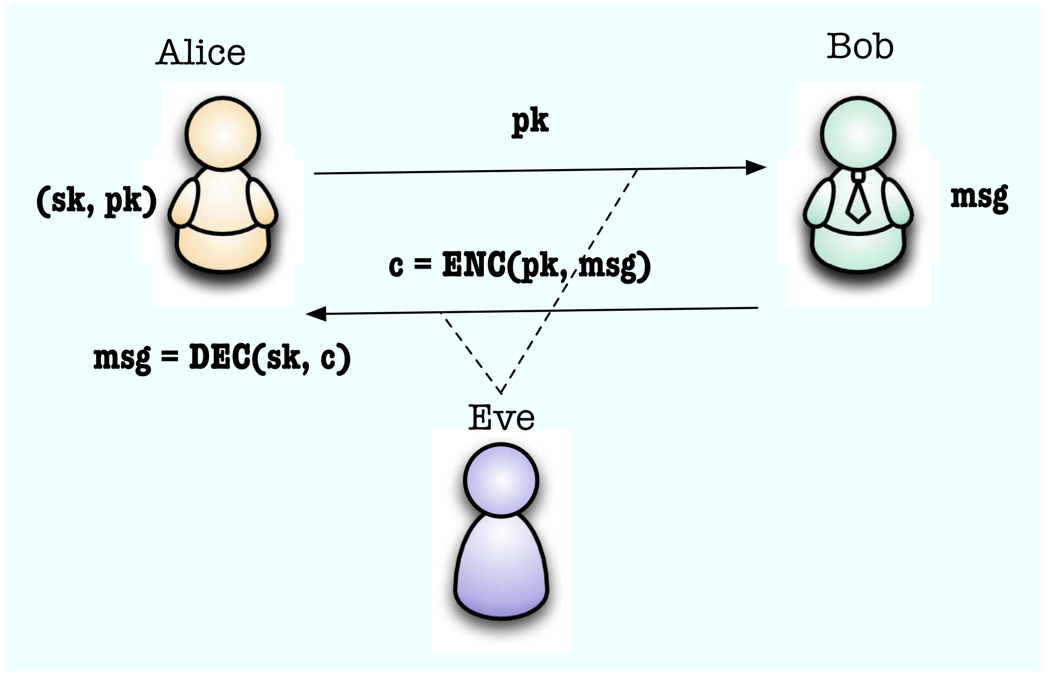
- What if you run out of keys?
- What if a key is stolen?

  "Why is it necessary to destroy yesterday's [key] . . . list if it's never going to be used again?"

  "A used key, Your Honor, is the most critical key there is. If anyone can gain access to that, they can read your communications."

  (trial of Jerry Whitworth, a convicted spy.)
- What if Alice doesn't know in advance that she'll want to talk to Bob?

How can Alice and Bob communicate?



ENC algorithm should be non-determnistic and use randomness.

The Solution: Public Key Cryptography

- Allows parties to communicate without prearrangement
- Separate keys for encryption and decryption
- Not possible to derive decryption key from encryption key
- Permissible to publish encryption key, so that anyone can send you secret messages
- All known public key systems are very expensive to use, in CPU time and bandwidth.
- Most public systems are based on mathematical problems.

Chosen Ciphertext Security

1. A key k is chosen running GEN.
2. **The adversary $\mathcal{A}$ can choose any text m and obtain ENC$_k(m)$. $\mathcal{A}$ can also choose any ciphertext c and obtain DEC(c).**
3. $\mathcal{A}$ chooses two messages m_0 and m_1.
4. A random $b \leftarrow \{0, 1\}$ is chosen. The ciphertext $c =$ ENC(m_b) is given to $\mathcal{A}$.
5. **In CCA1 security the adversary $\mathcal{A}$ can choose any text p and obtain ENC$_k$(m). In CCA2 security $\mathcal{A}$ can further choose any ciphertext $c' \neq c$ and obtain DEC(c').**
6. $\mathcal{A}$ outputs a bit b'.
7. The output of the experiment is $Priv_{\mathcal{A}}^{cca} = 1$ if $b = b'$ and $Priv_{\mathcal{A}}^{cca} = 0$ if $b \neq b'$.

Public Key Encryption

(GEN, ENC, DEC)

- GEN(security parameter) $\longrightarrow$ (pk, sk).
- ENC(pk, msg, rand) $\longrightarrow$ c

  Note: rand should be chosen randomly at each encryption
- DEC(sk, c) $\longrightarrow$ msg

  Note: DEC has to be deterministic

“Textbook RSA” and its Insecurity

- GEN(security parameter) $\rightarrow$ pk $= (N, e)$, sk $= (N, d)$
  $N = p \cdot q$ where p and q are large primes;
  $e \cdot d = 1 \bmod (p-1)(q-1)$
- ENC (pk, m) $\rightarrow c = m^e \bmod N$
- DEC (sk, c) $\rightarrow m = c^d \bmod N$

Attacks on “Textbook RSA”

- If e and msg have small such that $msg < N^{\frac{1}{e}}$, decryption is possible without d.
- If we fix $e = 3$ and encrypt the same message under three different keys $(N_1, e), (N_2, e), (N_3, e)$ we can recover the message from the three ciphertexts.
- Common modulus attack: several keys pk$_i$ = (N, e_i) and sk$_i$ = (N, d_i).
  - Each person having one key pair can read anything encrypted with any of the public keys.
  - If the same message is encrypted with two of these keys, an adversary who sees the two ciphertexts and knows the corresponding public keys can recover the message.
- Deterministic encryption, hence if we have a small range for message values, we can try each possible message.

Padded RSA

- GEN(security parameter) $\rightarrow$ pk $= (N, e)$, sk $= (N, d)$
- ENC(pk, m) $\rightarrow c = (rand||m)^e$ mod N where $rand$ is a random string of some fixed length l.
- DEC(sk, c) $\rightarrow \tilde{m} = c^d$ mod N removing the higher l bits.

Various padding schemes: PKCS (Public Key Cryptography Standard), OAEP (Optimal Asymmetric Encryption Padding), SAEP(Simple-OAEP)

A More Realistic Scenario

RSA operations are computationally expensive especially with long messages. A way to limit the length of the encrypted message:

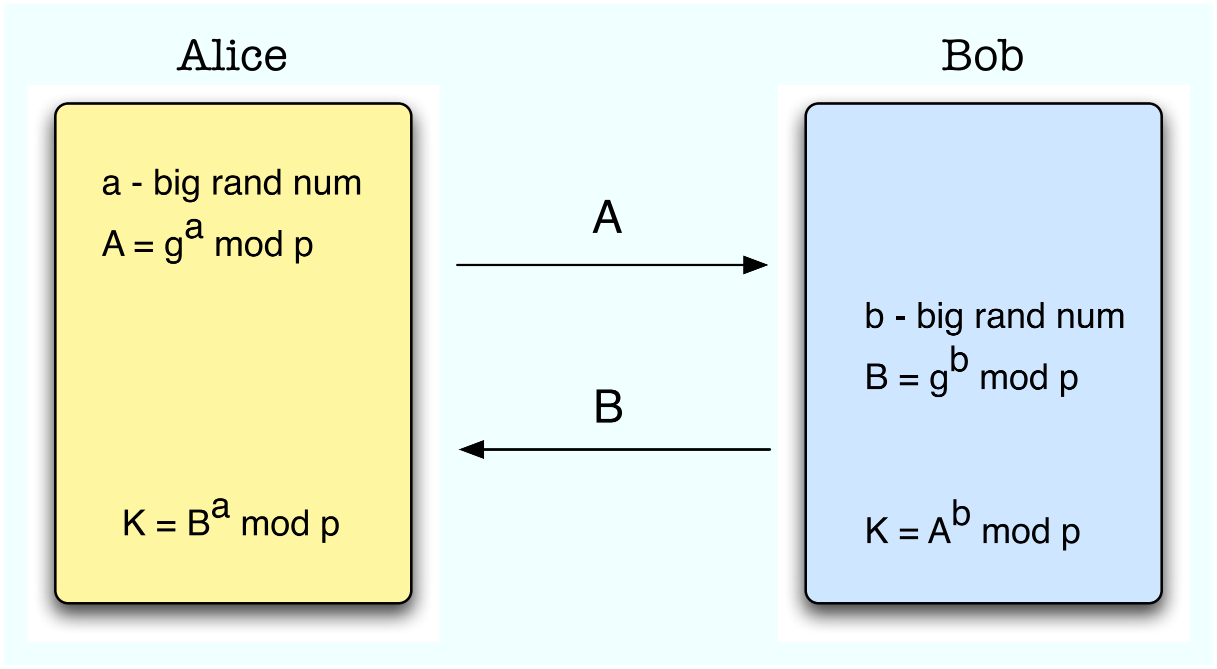
- Bob generates a random key k for a conventional cipher.
- Bob encrypts the message: $c = \{m\}_k$.
- Bob pads k with a known amount of padding, to make it at least 512 bits long; call this k'.
- k' is encrypted with Alice's public key $pk = (N, e)$.
- Bob transmits $\{c, ENC(pk, k')\}$ to Alice.
- Alice uses (N, d) to recover k', removes the padding, and uses k to decrypt ciphertext c.

Key Exchange

- Alice and Bob do not know each other and want to communicate.
- They need to establish a key that:
  - Nobody else knows.
  - The key cannot be recovered if the secret of one of them that was used in the key exchange protocol is revealed.

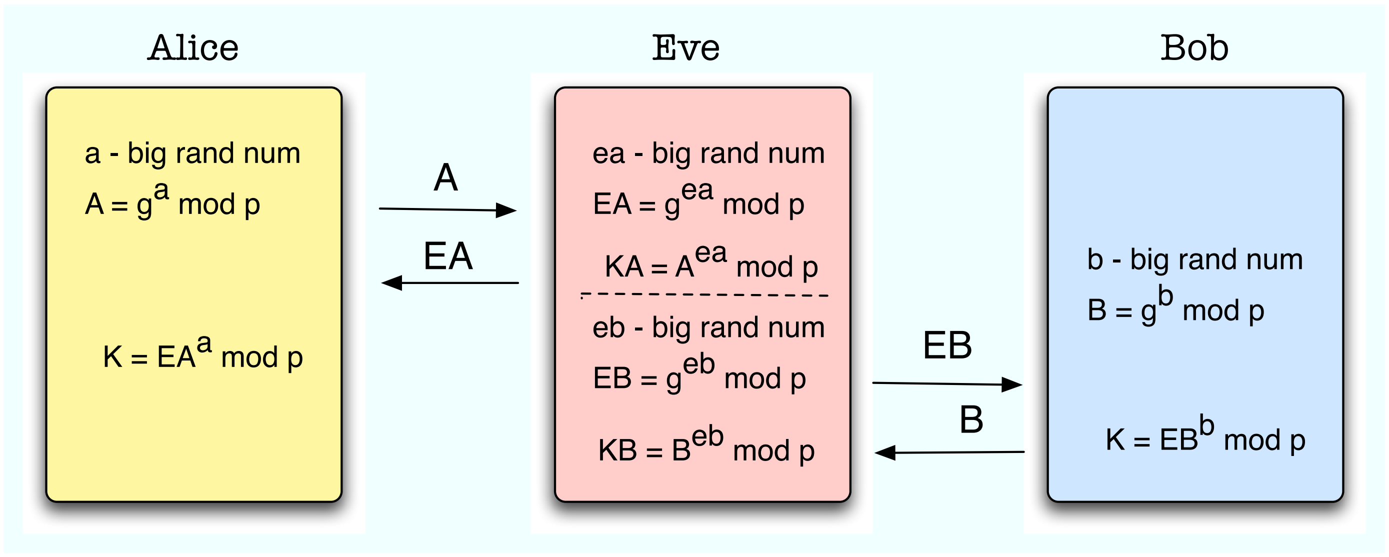
Diffie-Hellman Key Exchange

p is a large prime (usually $p = 2q + 1$ where q is prime), g is a generator for $\mathbb{Z}_p$



Discrete logarithm problem: given g and $g^x \bmod p$, cannot compute x.

Man-in-the-middle Attack



Authentication – encrypt with public key g^a, g^b, or K.

Hardness Assumptions

- *Factoring Hardness Assumption:* Factoring a number of the form $N = p \cdot q$, where p and q are big prime numbers, is hard.
- *RSA Hardness Assumption:* Let (N, e) and (N, d) be RSA keys. Given $y \in \mathbb{Z}_N^*$, it is hard to compute x such that $x^e = y \bmod N$.
- *Discrete Log Hardness Assumption:* Given $g, g^x \bmod p$, it is hard to compute x.
- *Decisional Diffie-Hellman (DDH) Hardness Assumption:* It is hard to distinguish tuples from type (g, g^x, g^y, g^{xy}) and type (g, g^x, g^y, g^z)

Forward Security

- If the current encryption keys are compomised, past encryptions cannot be compromised (read an old message, create a message and claim that it was sent in the past).

Homomorphic Vs. Non-malleable

- Homomorphic encryption (GEN, ENC, DEC) has the property that ENC(x + y) = ENC(x) + ENC(y). (voting schemes)
- Non-malleable encryption (GEN, ENC, DEC): given any number of encryptions ENC(y_i) you cannot obtain an encryption of ENC(x) where $x \neq y_i$ for all i. (bidding schemes)

Who Sent a Message?

- When Bob receives a message from Alice, how does he know who sent it?
- With traditional, symmetric ciphers, he may know that Alice has the only other copy of the key; with public key, he doesn't even know that
- Even if he knows, can he prove to a third party — say, a judge — that Alice sent a particular message?

Digital Signatures

(Gen, Sign, Vrfy)

- Gen(security parameter) $\longrightarrow$ (pk, sk).
- Sign(sk, msg) $\longrightarrow \sigma$
- Vrfy(pk, msg, σ) $\longrightarrow \begin{cases} 1, & \text{if the signature is valid} \\ 0, & \text{otherwise} \end{cases}$

  For every m it holds Vrfy(pk, m, Sign(sk, m)) = 1.

Signature Unforgeability

The signature experiment Sig-forge$_{\mathcal{A}}$:

- Run Gen to generate keys (pk, sk).
- The adversary $\mathcal{A}$ is given random oracle access to Sign – it submits a message and gets back its signature. $\mathcal{A}$ chooses a message m that has not been asked yet and outputs (m, σ).
- Sig-forge$_{\mathcal{A}}$ = $\begin{cases} 1, & \text{if Vrfy(pk, } m, \sigma) = 1 \\ 0, & \text{otherwise} \end{cases}$

The signature scheme is existentially unforgeable if for all probabilistic polynomial time adversaries: Pr[Sig-forge$_{\mathcal{A}}$ = 1] $\leq$ negl.

“Textbook RSA” Signature Scheme

- Gen(security parameter) $\longrightarrow$ pk = (N, e), sk = (N, d)
- Sign(sk, m) = m^d mod N.
- Vrfy(pk, m, σ) $\longrightarrow \begin{cases} 1, & \text{if } m = \sigma^e \bmod N \\ 0, & \text{otherwise} \end{cases}$

Attacks on “Textbook RSA” Signature Scheme

- Choose σ, compute $m = \sigma^e$ and output the valid signature pair (m, σ).
- If you have the valid signature pairs (m_1, σ_1) and (m_2, σ_2), you can compute the signature of $m_1 m_2$ as $(m_1 m_2)^d = m_1^d m_2^d = \sigma_1 \sigma_2$, hence $(m_1 m_2, \sigma_1 \sigma_2)$.

Solution: Hash before exponentiation: Sign(sk, m) = $(H(m))^d \bmod N$.

They’re Not Like Real Signatures

- Real signatures are strongly bound to the person, and weakly bound to the data
- Digital signatures are strongly bound to the data, and weakly bound to the person — what if the key is stolen (or deliberately leaked)?
- A better term: digital signature algorithms provide *non-repudiation*

Cryptographic Hash Functions

- Produce relatively-short, fixed-length output string from arbitrarily long input.
- Computationally infeasible to find two different input strings that hash to the same value
- Computationally infeasible to find any input string that hashes to a given value
- Strength roughly equal to half the output length
- Best-known cryptographic hash functions: MD5 (128 bits), SHA-1 (160 bits), SHA-256/384/512 (256/384/512 bits)
- 128 bits and shorter are not very secure for general usage

Hash Functions Security Properties

- Preimage resistance: given h, it is hard to find m such that $Hash(m) = h$.
- Second preimage resistance: given m_1, it is hard to find m_2 such that $Hash(m_1) = Hash(m_2)$.
- Collision resistance: It is hard to find two messages m_1 and m_2 such that $Hash(m_1) = Hash(m_2)$.

The Birthday Paradox

- How many people need to be in a room for the probability that two will have the same birthday to be $> .5$?
- Naive answer: 183
- Correct answer: 23
- The question is not “who has the same birthday as Alice?”; it’s “who has the same birthday as Alice or Bob or Carol or . . . ”, assuming that none of them have the same birthday as any of the others

Abusing a Weak Hash Function

- Alice prepares two contracts, m and m', such that $H(m) = H(m')$
- Contract m is favorable to Bob; contract m' is favorable to Alice

☞ The exact terms aren't important; Alice can prepare many different contracts while searching for two suitable ones.

- Alice sends m to Bob; he signs it, producing $\{H(m)\}_{K_B{}^{-1}}$.
- Alice shows m' and $\{H(m)\}_{K_B{}^{-1}}$ to the judge and asks that m' be enforced
- Note that the signature matches…

Attacks on Existing Hash Functions

- MD5
  - 1993, Den Boer and Bosselaers, ”pseudo-collision” of the MD5 compression function – two different IVs which produce an identical digest
  - 1996, Dobbertin, collision of the compression function of MD5
  - 2004, Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu, collisions for the full MD5
  - 2005, Arjen Lenstra, Xiaoyun Wang, and Benne de Weger, construction of two X.509 certificates with different public keys and the same MD5 hash
  - 2008, Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, Benne de Wege, creating rogue CA certificates

Attacks on Existing Hash Functions

- SHA-1
  - 2005, Rijmen and Oswald, attack on a reduced version of SHA-1 on 53 out of 80 rounds
  - 17 August 2005, Xiaoyun Wang, Andrew Yao and Frances Yao, an improvement on the SHA-1 attack, lowering the complexity required for finding a collision in SHA-1 to 2^{63}
  - 2006, Christian Rechberger, attack with 2^{35} compression function evaluations
- SHA-3 ?, NIST competition for new hash function http://csrc.nist.gov/groups/ST/hash/timeline.html

Message Authentication Codes (MACs)

(Gen, Mac, Vrfy)

- Gen(security parameter) $\longrightarrow k$; k must be kept secret
- Mac(k, m) $\longrightarrow t$.
- Vrfy(k, m, t) $\longrightarrow \begin{cases} 1, & \text{if the tag is valid} \\ 0, & \text{otherwise} \end{cases}$

  It holds that Vrfy(k, m, Mac(k, m)) = 1.

HMAC

- Let H be a hash function.
- $opad = 0x5c5c5c...5c5c$
- $ipad = 0x363636...3636$

$$HMAC_K(m) = H((K \oplus opad)||H((k \oplus ipad)||m))$$

Note: Message is appended not prepended.

Ellipctic Curve Cryptography (ECC)

- Elliptic curve

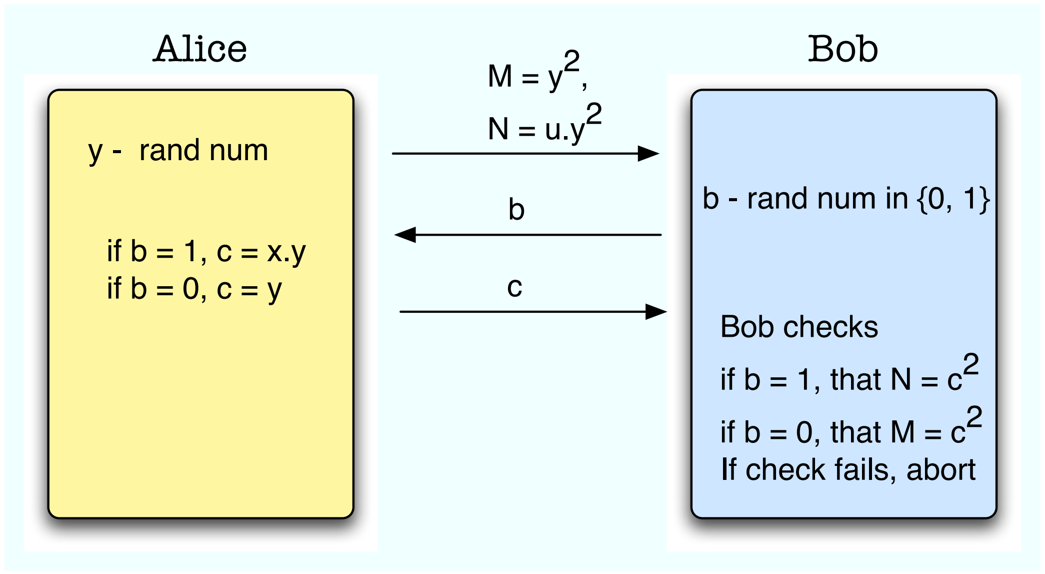
$$y^2 = x^3 + ax + b$$

- The solutions of an elliptic curve plus the point of infinity form a group in which the discrete log problem is believed to be much harder.
- There are no algorithms (yet) that break ECC assumptions in less than exponential time. This allows using shorter keys and better performance.
- Many patent issues that limit the use of ECC. (Certicom)

Zero Knowledge Proofs

How to prove that you know something without giving it away?

Example: *Alice wants to prove to Bob that she knows the square root of some big number* u*, i.e., she knows* x *such that* $x^2 = u$.
Proof: Run the following protocol n times:



Zero Knowledge Proofs

- If Alice can answer both type of changes, then she knows x such that $x^2 = u$ since $x = xy/y$.
- If the protocol completes then Alice can be cheating with probability at most $\frac{1}{n}$.

Applications:

- Prove that you know a secret that you want to sell without actually revealing it before the deal but convincing the potential buyer of your knowledge.
- Prove that the encryptions that you send have a certain property, e.g. lie on a polynomial of certain degree and uniquely reconstruct a value, encrypt the same values, etc.