Introduction to Modern Cryptography

Benny Chor

RSA Public Key Encryption
Factoring Algorithms

Lecture 7

Tel-Aviv University

Revised Dec. 19, 2009

Reminder: The Prime Number Theorem

- Let $\pi(x)$ denote the number of primes, p, up to x. For example, $\pi(30) = 4 + 4 + 2 = 10$.
- The prime number theorem: $\pi(x) \approx \frac{x}{\ln x}$.
  Furthermore, for $x \geq 55$,

$$\frac{x}{\ln x + 2} \leq \pi(x) \leq \frac{x}{\ln x - 4} \ .$$

- This means that there are a lot of primes. In particular, if we pick n bit long integers at random, then about one in n will be prime.

Reminder: Primality Testing $\in$ P

In terms of complexity classes, the Miller-Rabin algorithm, as well as the Solovay-Strassen algorithm, imply

Composites $\in$ RP

Where RP=Random Poly Time, one sided error.
Easy fact: RP is contained in NP.

The 2003 result of Agrawal, Kayal and Saxena is

Composites $\in$ P

For all practical purposes, the Miller-Rabin algorithm (and various optimizations thereof) supply a satisfactory and fast solution for identifying primes.

Integer Multiplication & Factoring as a One Way Function.

Multiplying two n bit numbers takes time $O(n^2)$.
Finding a random n bit long prime, and verifying its primality, is easy.
Factoring an n bit number takes time $2^{c \cdot n^{1/3}}$ (using the currently best algorithm).

Easy: $p, q \longrightarrow m = p \cdot q$ (integer multiplication).

Hard: $m = p \cdot q \longrightarrow p, q$ (integer factorization).

Question: Can public key cryptosystem be based on this observation?

The Multiplicative Group Z^*_{pq}

- Let p, q be two large primes.
- Denote their product $m = p \cdot q$.
- The multiplicative group $Z^*_m = Z^*_{pq}$ contains all integers in the range $[1 \ldots pq - 1]$ that are relatively prime to both p and q.

- The size of the group is $\phi(pq) = (p-1) \cdot (q-1) = m - (p+q) + 1$.
- So for every Z^*_{pq}, $x^{(p-1)\cdot(q-1)} = 1 \pmod{pq}$.

- Remark: Recall that Z^*_{pq} has no multiplicative generators.

Square Roots in Z^*_{pq}

Start with square roots of 1.

- in Z^*_p, 1 has two square roots: 1 and $p-1$.
- in Z^*_q, 1 has two square roots: 1 and $q-1$.
- What about the square roots of 1 in Z^*_{pq}?

- $y^2 = 1 \pmod{pq}$ iff $y^2 = 1 \pmod{p}$ and $y^2 = 1 \pmod{q}$.
- So $y = \pm 1 \pmod{p}$ and $y = \pm 1 \pmod{q}$.
- This gives rise to four systems of modular equations
  1. $y = 1 \pmod{p}$ and $y = 1 \pmod{q}$.
  2. $y = -1 \pmod{p}$ and $y = -1 \pmod{q}$.
  3. $y = 1 \pmod{p}$ and $y = -1 \pmod{q}$.
  4. $y = -1 \pmod{p}$ and $y = 1 \pmod{q}$.
- The solution to (1) is $y = 1$.
- The solution to (2) is $y = pq - 1 = -1 \pmod{p}$.
- The solutions to (3) and (4) are obtained using the Chinese remainder theorem.

Square Roots in Z_{pq}^*

In general, the square roots of y^2 are any of the four square roots of $1 \pmod{pq}$, multiplied by y.

- This implies that every square in Z_{pq}^* has exactly four square roots.
- The number of quadratic residues in Z_{pq}^* is $(p-1)(q-1)/4$.
- The mapping $x \rightarrow x^2 \mod pq$ is a four to one mapping.

Exponentiation in Z^*_{pq}

- Motivation: We want to use exponentiation for encryption.
- Let e be an integer, $1 < e < (p-1)\cdot(q-1)$.
- Question: When is exponentiation to the e-th power, $x \longrightarrow x^e$, a one-to-one operation in Z^*_{pq}?

Exponentiation in Z^*_{pq}

- Claim: If e is relatively prime to $\phi(m) = (p-1)\cdot(q-1)$, then $x \longrightarrow x^e$, a one-to-one operation in Z^*_{pq}.
- Constructive Proof: Since $\gcd(e, (p-1)\cdot(q-1)) = 1$, e has a multiplicative inverse mod $(p-1)\cdot(q-1)$. Denote this inverse by d, then $ed = 1 + C(p-1)(q-1)$, where C is an integer.
- Denote $y = x^e$, then $y^d = (x^e)^d = x^{ed} = x^{1+C(p-1)(q-1)} = x$. This means that $y \rightarrow y^d$ is the inverse operation of $x \rightarrow x^e$. ♠

The RSA Public Key Cryptosystem (reminder)

- Bob's private information: two large primes p, q.
- Public information: Their product, $m = p \cdot q$. An integer e that is relatively prime to $\phi(m) = (p-1) \cdot (q-1)$.
- More private information: An integer d that is relatively prime to $\phi(m) = (p-1) \cdot (q-1)$ and satisfies $d \cdot e = 1 \mod \phi(m)$.

- Messages P are elements in Z_m, namely numbers in $[1, \ldots, m-1]$.
- To encrypt P, compute $C = P^e \pmod m$, and send C to Bob.
- To decrypt C, Bob computes $C^d = P^{d \cdot e} = P \pmod m$.

"The above mentioned method should not be confused with the exponentiation technique presented by Diffie and Hellman to solve the key distribution problem"

The RSA Public Key Cryptosystem (MIT, 1978)

Rivest Shamir Adelman

Rivest (still at MIT), Shamir (now at Weizmann Inst.), and Adelman (now at USC), some 30 years ago.

Constructing an Instance of RSA Public Key Cryptosystem

- Bob picks at random two large primes p, q.
- Bob picks an integer e that is relatively prime to $\phi(m) = (p-1) \cdot (q-1)$. This does not have to be random or large.
- Bob computes d such that $ed = 1 \pmod{(p-1)(q-1)}$.

- Let $m = p \cdot q$ be the product of both primes.
- Bob publishes the public key m, e.
- Bob keeps the private key d, as well as the primes p, q and the number $\phi(m)$ in a safe place.

A Small Example

- Let $p = 47$, $q = 59$, $N = pq = 2773$. Then $\phi(N) = 46 * 58 = 2668$.
- Pick $e = 17$, then $157 \cdot 17 - 2668 = 1$, so $d = 157$ is the inverse of $e = 17$ mod 2668.
- $e = 17$ is 10001 in binary.
- For $N = 2773$ we can encode two letters per block, using a two digit number per letter: blank=00, A=01, B=02,. . . ,Z=26.
- The message: `ITS ALL GREEK TO ME` is encoded as
  `0920 1900 0112 1200 0718 0505 1100 2015 0013 0500`

A Small Example (cont.)

- $N = 2773$, $e = 17$ (10001 in binary).
- The message: `ITS ALL GREEK TO ME` is encoded as
  `0920 1900 0112 1200 0718 0505 1100 2015 0013 0500`

- Message is encrypted block by block.
- First block is $P_1 = 0920$. It encrypts to
  $P_1^e = P_1^{17} = (((P_1^2)^2)^2)^2 \cdot P_1 = 948 \pmod{2773}$.
- The whole message (ten blocks) ids encrypted as
  `0948 2342 1084 1444 2663 2390 0778 0774 0219 1655`
  Indeed $C_1^d = 0948^{157} = 920 \pmod{2773}$, etc.

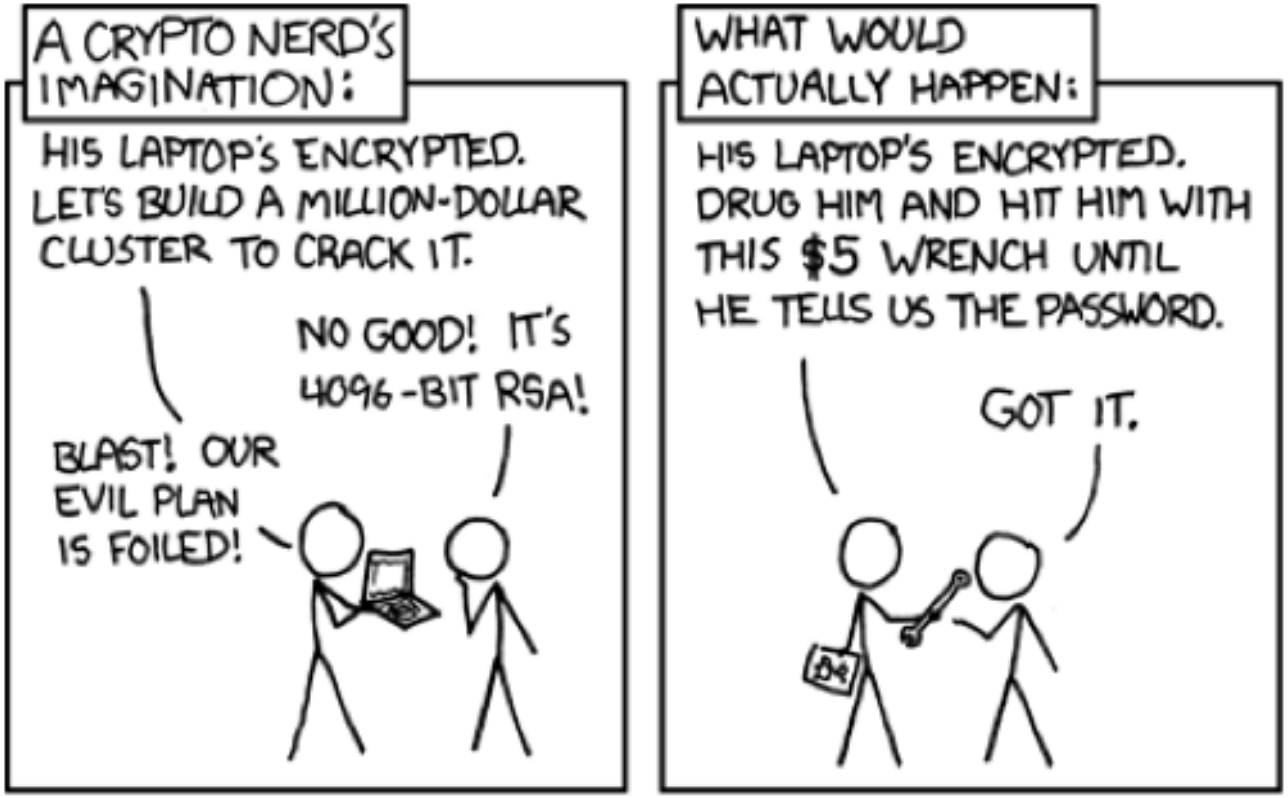
A A Slightly Larger Example

- Let $p = 2^{271} + 855$, $q = 2^{273} + 5$, $N = pq =$ 5758609657015291369997489289838056779353212314264532903689671329431521032595057735476212721821341837060063575156440993208752824217085409959745236008778839218983091. Then $\phi(N) = (p-1)(q-1) = (2^{271} + 854) \cdot (2^{273} + 4)$.
- Pick $e = 8777777777777776771$, $d = e^{-1}$ mod $(p-1)(q-1) =$ 3564847086250625759380533136146717837452553039130358724141242904052878601519619257740270715700985575943375150842891434724932654629619488829289568992534583837974508.
- For Message$=$ 4758609657015291369997489289838056779353212314264532903689671329431521032595057735476212721821341837060063575156440993208752824217085409959745236008 7 we can compute and verify, using *e.g.* Sage, that for Cipher$=$Message^e mod N, indeed Cipher^d mod $N$$=$Message.
- It is worthwhile doing similar computations by yourselves.

How "Strong" is RSA

- If Eve could factor pq then she obtains all private information of Bob.
- Hence breaking RSA cannot be harder than factoring.
- How hard is it to compute the secret key, d, from the public information m, e?
- If Eve can find d, then she can compute $ed - 1$. This is a multiple of $\phi(m) = (p-1)(q-1)$.
- Gary Miller has proved that such multiple enables factoring m. Rani Hod showed this in the recitation as well.
- Does this imply that "breaking" RSA is equivalent to factoring?

How "Strong" is RSA (the down-to-earth version*)



Original in `http://xkcd.com/538/`

*thanks to Elad Liebman for pointing this out

Some RSA Properties

- Deterministic Encryption:
  - Plaintext block P is always encrypted as $P^e \pmod{m}$.
  - So, just like ECB mode encryption of symmetric ciphers, we can detect repetitions.

- RSA is multiplicative (aka homomorphic):
  - Let P_1, P_2 be two plaintext blocks.
  - Then $E(P_1 \cdot P_2) = E(P_1) \cdot E(P_2)$.
  - This immediately implies that RSA does not behave like a pseudo random function (these are not multiplicative).
  - Multiplicativity implies volnurability to chosen ciphertext attacks.
  - Given a ciphertext $C = P^e \pmod{m}$, choose at random R and compute $C' = CR^e \pmod{m}$, which is an encryption of PR, a random looking text.
  - A decryption of $C' = CR^e \pmod{m}$ will reveal P (after one division mod m).

Using CRT to Speed Up RSA Decryption

- CRT preprocessing (done once):
  - Given p, q, compute a, b such that $ap + bq = 1$.
  - Define $A = ap, B = bq$ (coefficients of Chinese remainder theorem for p and q).

- Decryption, on input C:
  - Originally, Bob computed $C^d = P^{d \cdot e} = P \pmod{m}$.

  - Now, Bob computes $y_1 = C^{d \pmod{p-1}} \pmod{p}$.
  - And also $y_2 = C^{d \pmod{q-1}} \pmod{q}$.
  - Finally, Bob computes $P = Ay_1 + By_2 \pmod{m}$.
- This yields a non negligible saving (factor 4) in decryption runtime – essentially we deal with two half size exponentiations.

Using a Small Exponent to Speed Up RSA Encryption

- The public exponent, e, can be small
  - This will substantially improve efficiency of encryption.
  - It does not mean that the private exponent, d, is small too.
  - Neither does it seem to make it easier to derive d.
  - Small public exponents are used in practice.
  - Most popular values are either 3 or $2^{16}+1$.
  - The public exponent, e and $\phi(m) = (p-1)(q-1)$ should still be relatively prime.
  - For $e = 3$ this means 3 should divide neither $p-1$ nor $q-1$.

Using a Small Exponent to Speed Up RSA Encryption (2)

- Small exponents are fine for sending a single message to a single user, Bob.
- But they are vulnerable if the same message is to be sent to multiple users.
  - For example, suppose Bob, Bobi, and Bubchek are all using $e = 3$ (with public moduli m_1, m_2, m_3, respectively).
  - Alice wishes to send the plaintext message P to each of them.
  - She encrypts the plaintext message P and sends $P^3 \pmod{m_1}$, $P^3 \pmod{m_2}$, $P^3 \pmod{m_3}$.
  - Notice that $P < m_1, m_2, m_3$, and thus $P^3 < m_1 \cdot m_2 \cdot m_3$.
  - Denote $x = P^3$ (as an integer, not modulo anything).
  - Eve heard $x \pmod{m_1}$, $x \pmod{m_2}$, $x \pmod{m_3}$. She also knows that $x < m_1 \cdot m_2 \cdot m_3$.

Using a Small Exponent to Speed Up RSA Encryption (3)

- Eve heard $x \pmod{m_1}$, $x \pmod{m_2}$, $x \pmod{m_3}$. She also knows that $x < m_1 \cdot m_2 \cdot m_3$.
- m_1, m_2, m_3 are pairwise relatively prime.
- Eve (who overheard our Friday class on CRT and followed every iota) knows that this enables her to efficiently compute $y = x \pmod{m_1 m_2 m_3}$.
- But since $x < m_1 \cdot m_2 \cdot m_3$, this implies that $y = x$ (over the integers!). Therefore Eve now holds $P^3 = x$ without any modular subtractions.
- Computing cubic roots over the integers is easy.
- So Eve finds P and successfully retrieves the plaintext.

- Is this a weakness of small exponent RSA? No!
- It is a weakness in the manner small exponent RSA was used.
- In general, if we have cryptographic building blocks that are secure on their own, this does not mean we can compose or interleave them any way we want, and still maintain security.

Random Self Reducibility of RSA

- Let $m = pq, e$ be a given RSA modulus, of length n bits, plus a RSA exponent.
- Suppose $\mathcal{A}$ is an deterministic n^c time algorithm which, on input $E(x) = x^e \pmod m$, outputs x for εm of the inputs in Z_m.
- Then $\mathcal{A}$ can be converted into a randomized $\varepsilon^{-1} n^c$ expected time algorithm, $\mathcal{R}$ which, on input $E(x) = x^e \pmod m$, outputs x for all the inputs in Z_m.
- Whenever $\mathcal{R}$ terminates, it gives the correct output x.

- Proof: on board (easy).
- Consequence: For a given $m = pq$ and e, inverting RSA is either hard everywhere or easy everywhere.

Random Padding (aka "Salting") of RSA

- Padding the message by a block of random bits: Suppose the length of pq is n bits. Use ℓ bits for the message P, concatenate with $n - \ell$ random bits string, r: $E(r \circ P) = (r \circ P)^e \pmod{pq}$.
- Padding reduces the information rate, but increases security. It can be shown that if $n - \ell$ is very large, then padded RSA is resistant to chosen plaintext attack.
- Of course for security to hold, pad must be random. Choosing $r = \texttt{hello world}$, or any other fixed text, is not a good practice.
- For protection against chosen ciphertext attack, a combination of fixed and random padding was proposed by RSA labs: Let P be a ℓ bit long message. Pad and encrypt by $(\texttt{00000000} \circ \texttt{00000010} \circ r \circ \texttt{00000000} \circ P)^e \pmod{pq}$.
- Fixed parts of pad intended to foil multiplication attacks.
- Unfortunately, some chosen ciphertext attacks were later found. Still, scheme is being used.

General Remark on Public Key Cryptosystems

- PKCs are order of magnitude slower than private key systems. Hence used mainly to exchange keys or signing.
- Under suitable complexity assumptions, PKC are secure, provided we can trust the association of keys with users.
- If I were tricked to send a message using what I think is the public key of Esau, but Jacob (a well known trickster) is the one that can decipher it, then I may be in trouble.

Isaac rejecting Esau, by Giotto di Bondone, 13-14th centuries, Assissi, Italy.

- To achieve secure communication without prior physical contact, have to establish (and trust) centers for distributing certificates.
- Will be discussed (soon) under "public key infrastructure".

Real World Usage of RSA

(1) Key exchange.

(2) Digital signatures.

Signatures

http://lacourphoto.net/uploaded_images/signatures1-770492.jpg

Hand Written Signatures

- Relate an individual, through a handwritten signature, to a document.
- Signature can be verified against a prior authenticated one, which was signed in person in a bank, in the presence of a public notary public, etc.
- Should be hard to forge.
- Are legally binding (convince a third party, e.g. a judge).

Digital Signature Schemes

- Relate an individual, through a digital string, to a document.
- Would like to achieve all features of hand written signatures, plus more.
- For example, should be able to base difficulty of forgery on some hard computational problem, not just on ineptitude of forger.
- Diffie and Hellman were first to propose such framework.
- An implementation was first given by RSA.

Diffie and Hellman "New Directions in Cryptography" (76)

Diffie and Hellman proposed a "textbook framework", based on any deterministic public key cryptosystem.

We will describe this framework and implementation(s), and then discuss some specific shortcomings of it.

- Let E_A be Alice's public encryption key, and let D_A be Alice's private decryption key.
- To sign the message M, Alice computes the string $y = D_A(M)$ and sends (M, y) to Bob.
- To verify this is indeed Alice's signature, Bob computes the string $x = E_A(y)$ and checks that indeed $x = M$.

Diffie and Hellman "New Directions in Cryptography" (76)

Diffie and Hellman proposed a "textbook framework", based on any deterministic public key cryptosystem.

We will describe this framework and some implementation(s), and then discuss some specific shortcomings of it.

Intuition: Only Alice can efficiently compute $y = D_A(M)$, thus forgery should be computationally infeasible.

Question: Do you buy this argument?

Problems with "Pure" Diffie-Hellman Paradigm

- Easy to forge signatures of random messages, even without holding D_A:
- Bob picks R arbitrarily, and computes $s = E_A(R)$ using Alice public key.
- Then the pair (s, R) is a valid signature of Alice on the "message" s.
- Therefore the scheme is subject to existential forgery.
- "So what" ?

Problems with RSA Implementation

- Consider specifically RSA. Being multiplicative, we have (products are modulo pq): $D_A(M_1 M_2) = D_A(M_1) D_A(M_2)$.

- If $M_1 =$ "`I OWE BOB $20`" and $M_2 =$ "`100`" then under certain encodings of letters, we could get $M_1 M_2 =$ "`I OWE BOB $2000`" ...

Generic Solution: Hash First

- Let E_A be Alice's public encryption key, and let D_A be Alice's private decryption key.
- Let H be a strongly collision resistant hash function.
- The hash function H is completely public, and in particular no secret key is employed.
- To sign the message M, Alice first hashes the message, namely computes the string $y = H(M)$.
- Then, Alice computes the string $z = D_A(y)$. She sends (M, z) to Bob.
- To verify this is indeed Alices signature, Bob computes the string $y = E_A(z)$ and checks that indeed $y = H(M)$.

- Argue (intuitively) why this foils previous attacks.

Signature Schemes: General Structure

- Generation of private and public keys (this phase must use truly random bits).
- Signing algorithm: Either deterministic or randomized.
- Verification algorithm: Returns an accept/reject output. Usually deterministic.

- It is important to realize that signature schemes that deviate from the Diffie-Helman plus hashing paradigm do exist (*e.g.* Goldwasser-Micali-Rivest, 1988).

Signatures vs. MACs: Important Distinction

- The obvious difference is that MACs require shared secret keys, while signature keys do not.
- But there is a more important distinction:

- Suppose parties A and B share the secret key K.
- Then $(M, MAC_K(M))$ convinces A that indeed M originated with B.
- But in case of dispute, A cannot convince a judge that $(M, MAC_K(M))$ was sent by B , since A could have generated it herself.

RSA as a One Way Trapdoor Function

Easy: $x \longrightarrow x^e = y \pmod{pq}$ (e is known).

Hard: $y \longrightarrow y^d = x \pmod{pq}$ (d is unknown).

Easy with trapdoor information: $y \longrightarrow y^d = x \pmod{pq}$ (d is known).

Trapdoor One Way Functions

- Definition: $f : D \rightarrow R$ is a trapdoor one way function if there is a trapdoor s such that:
  - Without knowledge of s, the function f is a one way function.
  - Given s, inverting the function f is easy.
- Example: the function $f_{g,p}(x) = g^x \pmod{p}$ is not a trapdoor one way function.
- Example: RSA is a trapdoor one way function.