

# Introduction to Modern Cryptography

Benny Chor

Factoring Algorithms: Brief Overview  
Elgamal PKC, Based on Discrete Logs  
Digital Signatures (Reminder) Elgamal Signature, Based on  
Discrete Logs

Lecture 9

Tel-Aviv University

Modified 19 December, 2009

# Factoring Algorithms: Very Brief Overview

What is the running time (worst case) of factoring algorithms? Let  $m$  be an  $n$  bits composite. Hardest numbers to factor are the product of two distinct prime numbers  $m = pq$ , where both  $p - 1$  and  $q - 1$  have a large prime factor.

A (very partial) list of algorithms:

- Trial division:  $O(2^{n/2})$ .
- J.M. Pollard's rho method:  $O(2^{n/4})$  (discussed in lecture 7).
- Quadratic sieve algorithm:  $O(e^{(n \log n)^{1/2}})$  (discussed in recitation 8).
- General number sieve algorithm:  $O(e^{(7n)^{1/3} \cdot \log^2 n})$ .
- GNS was introduced by J.M. Pollard in 1988, and later refined by many well-known players of the computational number theory community.

# Factoring Algorithms

- The general number sieve algorithm is considered the fastest of all published, “general purpose” factoring algorithms. It was employed to factor **RSA-200**, a 663-bit number (200 decimal digits), on May 2005. The algorithm was implemented on a cluster of 80 2.2 GHz Opterons. Execution took three months.

- RSA-200 =  
2799783391122132787082946763872260162107044678695542853756000992932612840010760934567105295  
5360856061822351910951365788637105954482006576775098580557613579098734950144178863  
178946295187237869221823983  
Factors =  
3532461934402770121272604978198464368671197400197625023649303468776121253679423200058547956528088349  
and  
7925869954478333033347085841480059687737975857364219960734330341455767872818152135381409304740185467

This is still **very far** from factoring the 4096 bits RSA key, which the XKCD hackers were after.

- We have embarked upon a much more **modest task**: Explain Pollard’s **rho** method (on the board), implement it on a Sage web server, and run it to factor an 100-bit number (in approximately two minutes).

# Elgamal Public Key Cryptosystem

- We are now going to describe a second PKC, designed by Taher Elgamal in 1985 (he was with Netscape at the time).



- Elgammal PKC is based on the difficulty of finding discrete logs in finite fields, and more specifically on the Diffie and Hellman key exchange assumption.
- We will start by reviewing Diffie-Hellman, then move to Elgamal.

# Diffie and Hellman Key Exchange (reminder)

- **Public parameters:** A large prime  $p$  (1024 bits, say) and a primitive element  $g$  in  $Z_p^*$ .
- Alice chooses **at random** an integer  $a$  from the interval  $[0..p-2]$ . She sends  $x = g^a \pmod{p}$  to Bob (over the insecure channel).
- Bob chooses **at random** an integer  $b$  from the interval  $[0..p-2]$ . He sends  $y = g^b \pmod{p}$  to Alice (over the insecure channel).
- Alice, holding  $a$ , computes  $y^a = (g^b)^a = g^{ba}$ .
- Bob, holding  $b$ , computes  $x^b = (g^a)^b = g^{ba}$ .
- Now both have the **shared secret**,  $g^{ba}$ .

# Elgamal PKC (note resemblance to DH)

- Public information: A large prime  $p$ , where  $p - 1$  has a known factorization and a large prime factor. Recommended to take  $p = 2q + 1$ ,  $q$  is prime too, and  $p$  is 756 or 1024 bits long.
  - ▶ A multiplicative generator  $g$  of  $Z_p^*$
  - ▶ Bob publishes  $p, g$ .
  - ▶ Bob picks  $a \in [0..p - 2]$  at random.
  - ▶ Bob computes and publishes  $\beta = g^a \pmod{p}$ .
- Bob's private information:  $a$ .
- Encryption: of the message  $m$ :
  - ▶ Alice picks  $k \in [0..p - 2]$  at random.
  - ▶ Alice computes  $g^k \pmod{p}$ ,  $m\beta^k \pmod{p}$ .
  - ▶ Alice sends  $E(m) = (g^k, m \cdot \beta^k)$  to Bob.  
( $\beta^k$  "masks"  $m$ ;  $k$  obviously is not made public).
- Decryption of  $(g^k, m \cdot \beta^k) = (c_1, c_2)$ :
  - ▶ Bob computes  $c_1^a = (g^k)^a = (g^a)^k = \beta^k \pmod{p}$ .
  - ▶ This enables Bob to compute the multiplicative inverse of  $\beta^k \pmod{p}$ ,  $\beta^{-k}$  (even though he does not know  $k$ ).
  - ▶ Bob now computes  $\beta^{-k} \cdot c_2 = m$ . ♠

# Properties of Elgamal Public Key Cryptosystem

- Encryption is **randomized**:  $m \rightarrow (g^k, m\beta^k)$ .
- Alice should use a new, independent  $k$  for every encryption.
- Even if same  $m$  is sent twice, different  $k$  must be used.
  
- Encryption takes two modular exponentiations.
- Decryption takes one modular exponentiation.
- Ciphertext,  $(g^k, m\beta^k)$ , is twice as long as plaintext  $m$ .

## Properties of Elgamal Public Key Cryptosystem (2)

- Cryptosystem is **vulnerable** to chosen ciphertext attacks.
- Given  $E(m) = (c_1, c_2) = (g^k, m\beta^k)$ ,
- Attacker chooses a random  $s$ , computes  $(c_1, s \cdot c_2) = (g^k, s \cdot m\beta^k)$
- Attacker asks for decryption of  $(c_1, s \cdot c_2)$ , which equals  $s \cdot m$ , from which  $m$  is easily recovered.
  
- Cryptosystem is **multiplicative**. Given  $E(m) = (c_1, c_2) = (g^k, m\beta^k)$ ,  $E(m') = (c'_1, c'_2) = (g^{k'}, m'\beta^{k'})$ , can easily obtain  $E(m \cdot m') = (c_1 c'_1, c_2 c'_2) = (g^{k+k'}, m \cdot m' \beta^{k+k'})$  (without knowing any secret information).



# Does DH Key Exchange Hide All Partial Information? (reminder)

- From  $g^a$  and  $g^b$ , Eve could easily deduce if  $a$  and  $b$  are even or odd. The exponent arithmetic is done modulo  $p - 1$ , which is even.
- If both  $a$  and  $b$  are odd, then  $ab \pmod{p - 1}$  is odd too, and  $g^{ba}$  is not a QR. If  $a$ ,  $b$ , or both are even, then  $ab \pmod{p - 1}$  is even, so  $g^{ba}$  is a QR.
- Thus in (this original version) of DH key exchange, does leak some partial information – specifically the QR bit of the key  $g^{ba}$ .
- Same type of partial information is leaked in Elgammal encryption.

# Does Elgamal Encryption Hides All Partial Information? (reminder)

- From  $\beta = g^a$ , Eve could easily deduce if  $a$  is even or odd.
- From  $g^k$ , Eve could easily deduce if  $k$  is even or odd.
- If both  $a$  and  $k$  are odd, then  $ak \pmod{p-1}$  is odd too, and  $\beta^k = g^{ak}$  is not a QR. If  $a$ ,  $b$ , or both are even, then  $ab \pmod{p-1}$  is even, so  $g^{ak}$  is a QR.
- Thus from  $m\beta^k = mg^{ak}$ , Eve can deduce if  $m$  is a QR or not a QR.
  
- So this type of partial information is leaked in Elgammal encryption as well.



## Restricting the Message Space

- Standard fix for DH key exchange to this partial information leakage problem:  $p$  is chosen to be of the form  $p = 2q + 1$ , where  $q$  is a prime.
- Instead of working in  $Z_p^*$ , work with QR, the quadratic residues of  $Z_p^*$ .
- QR is a cyclic group with exactly  $q$  elements.
- Instead of working with a multiplicative generator  $g$  of  $Z_p^*$ , work with a multiplicative generator  $h$  of QR, the quadratic residues of  $Z_p^*$ .
- An identical fix is applicable to Elgamal PKC.
- Alice should now encode messages as quadratic residues.
- Encoding messages as QR elements is easiest if  $-1$  is not a QR in  $Z_p^*$ . We omit the details.

# Signatures (Reminder)

James A. Ozama  
Frederick C. Manovich  
Daniel Kottke  
Benny Chor  
Charles Gagnier  
Bill Atkinson  
Nick Millidge  
Bruce Horn  
George Gass  
Rod Holt  
Andy Kitzfeld  
Angelia L  
Joanna Kaufman  
HAP HORN  
W.E. McCannnon  
Burrill Smith  
Jef Raskin  
Lloyd Hill III  
Roger Abiko  
Pamela Skop  
Steve Jobs  
Brian Rotstein  
Joan Marsh  
Larry Kenyon  
Martin P. Hales  
Cecilia Oakland  
Lamberto Pidel  
Mike Boich  
Kyrin Sakahachi  
Benjamin L. Perry  
Bill Fernandez  
Donna Demmer  
Tatti King  
David H. Roberts  
Ronald H. Whitehorn  
Matt Carter  
Robert J. Callenville  
Randy Wigginton  
Linda Wilkins  
Michael Manning

FEBRUARY 10, 1982

[http://lacourphoto.net/uploaded\\_images/signatures1-770492.jpg](http://lacourphoto.net/uploaded_images/signatures1-770492.jpg)

Introduction to Modern Cryptography Benny Chor

Factoring Algorithms: Brief Overview Elgamal PKC, Based on Discrete Logs Digital Signatures (Reminder) Elgamal Signature, Base

# Digital Signature Schemes (Reminder)

- 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**.
- Specific implementations may pose additional threats.
- For example, since RSA is multiplicative, we have (products are modulo  $pq$ ):  $D_A(M_1M_2) = D_A(M_1)D_A(M_2)$ .



## 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 Alice's 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).

# Signature Schemes Used in Practice

- RSA
- Elgamal **Signature** Scheme (1985), based on the intractability of discrete log in  $Z_p^*$ . Will be described shortly.
- The **DSS/DSA** (digital signature standard/algorithm), adopted by NIST in 1994. It is based on a modification of El-Gamal signature.
- Elgamal like scheme in the different groups of **elliptic curves**. This will probably **not** be described in this course.

# Elgamal Signature Scheme

- **Key Generation:**

A large prime  $p$  with  $p - 1$  having a known factorization and a large prime. Recommended to take  $p = 2q + 1$ , where  $q$  is also a prime, and  $p$  is 1024 bits long.

A standard, strongly collision resistant hash function,  $H$ .

- ▶ Bob picks a **multiplicative generator**  $g$  of  $Z_p^*$ .
  - ▶ Bob picks  $x \in [0..p - 2]$  **at random**.
  - ▶ Bob computes and publishes  $y = g^x \pmod{p}$ .
- Bob's private key is  $x$ .
  - Bob's public signature key is  $p, g, y$ .

So far it is similar to Elgamal encryption. This will soon change.

## Elgamal Signature Scheme (2)

Signing the message  $M$ , employing randomization:

- Hash: Let  $m = H(M)$ :
- Bob picks at random  $k \in [0..p-2]$  which is relatively prime to  $p-1$  (there are  $\varphi(p-1) \geq q-1$  such elements).
- Bob computes  $r = g^k \pmod{p}$ .
- Bob computes  $s = (m - rx) \cdot k^{-1} \pmod{p-1}$  (\*\*\*)
- Bob outputs  $r, s$ .
- Together with  $M$ , this is the signature of  $M$ .

## Elgamal Signature Scheme (3)

- Bob outputs  $r, s$ .
- Together with  $M$ , this is the signature of  $M$ .
- **Verification** of  $M$ 's signature:
- Alice computes  $m = H(M)$ .
- If  $0 < r < p$  and  $y^r r^s = g^m \pmod{p}$ , Alice **accepts**. Otherwise she **rejects**.
- What the  $\&\%\$ \#$  is going on?
- By (\*\*\*) ,  $s = (m - rx) \cdot k^{-1} \pmod{p-1}$ .  
So  $sk + rx = m \pmod{p-1}$ .
- By construction  $r = g^k$ , so  $r^s = g^{ks}$ .
- Since  $y = g^x$ , we have  $y^r = g^{rx}$ ,  
implying  $y^r r^s = g^{rx} g^{ks} = g^{sk+rx} = g^m$ .



## Elgamal Signature Scheme: Some Properties

- Since signing is randomized, same message could have multiple, different signatures. OK to sign same message many times.
- If  $k \in [0..p-2]$  is **repeated** in different mssgs (rather being chosen **at random**), then Eve can extract the private key (try this!).
- Same applies to highly dependent choices of  $k_1, k_2, \dots$
- Can be applied in other commutative groups where discrete log seems hard. For example in **Elliptic curves** (more efficient operations).
- Forging amounts to finding  $r, s$  such that  $y^r r^s = g^m \pmod{p}$ , and seems to require discrete logs (but not proved equivalent!)
- Overhead:
  - ▶ **Signature**: One exponentiation, can be done offline (preprocessing). A constant number of modular multiplications/gcd (cheap).
  - ▶ **Verification**: Three exponentiations, plus a constant number of modular multiplication.

## Signatures vs. MACs: Important Distinction (Reminder)

- 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.



## Quo vadis? – Eo Romam Iterum Crucifigi

Well, our planned future is not as bleak as the (claimed) source of this quote.

So far, we became familiar with a variety of **cryptographic primitives**, as well as specific implementations of them. These include:

- Pseudo random generators, pseudo random function and permutation generators (mostly intuitive notions),
- Encryptions (stream cyphers, block cyphers, different modes of operation) and MACs (in the symmetric key setting).
- Cryptographic hashing (no key).
- Public key framework; Key exchange (Diffie Hellman).
- Encryptions and signature schemes (in the public key setting).

## Quo vadis? – Eo Romam Iterum Crucifigi

Well, our planned future is not as bleak as the (claimed) source of this quote.

In the remaining weeks, we will introduce a number of additional **cryptographic primitives**, as well as cryptographic notions with proposed implementations. These include:

- Secret sharing.
- Zero knowledge proofs.
- Hard core bits.
- Coin flipping over the phone.
- Identification and user authentication schemes.
- Fingerprinting files.

## $n$ -out-of- $n$ Secret Sharing

A not so hallucinated scenario:

- A value  $S \in \mathcal{U}$  is the secret key allowing access or activation of an extremely critical and sensitive device.
- A “trusted dealer” holds  $S$ , but does not wish to activate the device right now.
- This dealer wants to delegate the secret to  $n$  parties.
- The parties can only be **partially trusted**: We seek a mechanism that will enable all  $n$  parties to reconstruct  $S$ , but any subset of  $n - 1$  parties (or less) cannot get **any partial information** on  $S$ .
- Computational requirements: Reconstruction should be efficient (as a function of length of  $S$  and of number of parties  $n$ ).
- Impossibility of reconstruction by  $n - 1$  parties should be **information theoretical**.

# $n$ -out-of- $n$ Secret Sharing

Simple solution detailed on board:

Express  $S = \sum_{i=1}^n r_i \pmod{|U|}$ .

## $t$ -out-of- $n$ Secret Sharing

Another, not so hallucinated scenario:

- A value  $S \in \mathcal{U}$  is the secret key allowing access or activation of an extremely critical and sensitive device.
- A “trusted dealer” holds  $S$ , but does not wish to activate the device right now.
- This dealer wants to delegate the secret to  $n$  parties.
- The parties can only be **partially trusted**: We seek a mechanism that will enable any  $t$  parties to reconstruct  $S$ , but any subset of  $t - 1$  parties (or less) cannot get **any partial information** on  $S$ .
- Computational requirements: Reconstruction should be efficient (as a function of length of  $S$  and of number of parties  $n$ ).
- Impossibility of reconstruction by  $t - 1$  parties should be **information theoretical**.

## $t$ -out-of- $n$ Secret Sharing

Slightly more complicated solution detailed on board:

Choose at random a degree  $t + 1$  polynomial  $f$ , over a finite field with at least  $\max\{|U|, n\}$  elements, whose free term equals  $S$ . The share given to party  $i$  ( $i = 1, 2, \dots, n$ ) is the value  $f(i)$ .