CSCI361
Computer Security

Public Key Cryptography IV
Rabin and Elgamal

Outline

- The Rabin cryptosystem.
  - Description.
  - An example.
  - Advantages and disadvantages.
- The Elgamal cryptosystem.
  - Description.
  - $\mathbf{Z}_p$: primitives/generators.
  - An example.

The Rabin cryptosystem

- The security of this system is equivalent to the difficulty of factoring.
- **Key generation**:
  - Bob randomly chooses two large primes,
    - **p** and **q**, and
    - calculates **N=pq**.
  - The public key is **N** and
    - The secret key is **(p,q)**.

- **To Encrypt**: the ciphertext **Y** for a message (plaintext) **X:**

  $Y=X^2 \bmod N$

  Note this is RSA with **e=2**!

- **To decrypt** the received message:
  - Bob must find the square root of Y mod N.
  - Knowing (p,q) it is easy to find the square root, otherwise it is provably as difficult as factoring.

- **Special case:** **p** and **q** are 3 mod 4.
  - We construct four intermediate factors.

    $\mathbf{x_1=Y^{(p+1)/4} \bmod p}$

    $\mathbf{x_2=p-x_1}$

    $\mathbf{x_3=Y^{(q+1)/4} \bmod q}$

    $\mathbf{x_4=q-x_3}$

- We define

  $a=q(q^{-1} \bmod p)$ **$b=p(p^{-1} \bmod q)$**

- This means, for example, that you calculate **q^{-1} mod p** and multiply the result by **q**.

- Then 4 **possible plaintexts** can be calculated.

$$X_1 = (ax_1+bx_3) \bmod N$$
$$X_2 = (ax_1+bx_4) \bmod N$$
$$X_3 = (ax_2+bx_3) \bmod N$$
$$X_4 = (ax_2+bx_4) \bmod N$$

An example

- We choose p=7, q=11 so N=77.
  - Note that p and q are 3 mod 4.

- Bob's public key is 77, and his private key is (7,11).

- To encrypt **X=3** Alice calculates

  $Y=3^2 \bmod N = 9 \bmod 77$.

- To decrypt Bob calculates

  $x_1=9^2 \bmod 7 = 4$ $\quad\quad$ $x_2=7-4=3$

  $x_3=9^3 \bmod 11 = 3$ $\quad\quad$ $x_4=11-3=8$

You can calculate $9^3 \bmod 11$ as $(-2)^3 \bmod 11=-8 \bmod 11=3$.

- Bob then finds **a** and **b**:

  $7(7^{-1} \bmod 11) = 7 \times 8 = 56$

  $11(11^{-1} \bmod 7) = 11 \times 2 = 22$

- … and then the four possible plaintexts.

  $X_1 = 4 \times 22 + 3 \times 56 = 11 + 14 = \mathbf{25} \bmod 77$

  $X_2 = 4 \times 22 + 8 \times 56 = 11 + (-14) = -3 = \mathbf{74} \bmod 77$

  $X_3 = 3 \times 22 + 3 \times 56 = -11 + 14 = \mathbf{3} \bmod 77$

  $X_4 = 3 \times 22 + 8 \times 56 = -11 - 14 = -25 = \mathbf{52} \bmod 77$

Advantages and disadvantages.

- Provable security.
- Unless the RSA exponent *e* is small, Rabin's encryption is considerably faster than RSA, requiring **one modular exponentiation**.
  - Decryption requires roughly the same time as RSA.

- Decryption of a message generates 4 possible plaintexts. The receiver needs to be able to decide which one is the right message. One can append messages with known patterns, for example 20 zeros, to allow easy recognition of the correct plaintext.

- Rabin's system is mainly used for authentication (signatures).

Generator of Z_p^*

- An element α is a generator of Z_p^* if
  α^i, $0 < i \leq p-1$ generates all numbers *1,.. p-1*
- Finding a generator in general is a hard problem with no efficient algorithm known.
- If factorization of *p-1* is known it is not hard.
- In particular if $p=2p_1+1$ where p_1 is also a prime we have the following.

- α in Z_p^* and α ≠ ± 1 mod p.
- Then α is a primitive element if and only if

$$\alpha^{\frac{p-1}{2}} \neq 1 \bmod p$$

- Suppose α in Z_p^* and α is not a primitive element.
- Then $-\alpha$ is a primitive element.

- This gives an efficient algorithm to find a primitive elements.

- Example: Z^*_{11}
  - $3^5 = 1 \bmod 11$
  - 3 is not primitive, -3 = 8 is primitive

An example :Z_{11}^*

	1	2	3	4	5	6	7	8	9	10	
1	1	1									
2	**2**	**4**	**8**	**5**	**10**	**9**	**7**	**3**	**6**	**1**	
3	3	9	5	4	1	3					
4	4	5	9	3	1	4					
5	5	3	4	9	1	5					
6	**6**	**3**	**7**	**9**	**10**	**5**	**8**	**4**	**2**	**1**	
7	**7**	**5**	**2**	**3**	**10**	**4**	**6**	**9**	**8**	**1**	
8	**8**	**9**	**6**	**4**	**10**	**3**	**2**	**5**	**7**	**1**	
9	9	4	3	5	1	9					
10	10	1	10								

Discrete Logarithm Problem (DLP)

INPUT:

- Z_p^*
- g *in* Z_p^* , g a generator of Z_p^*
- h in Z_p^*

Find the unique number $a<p$ such that $h=g^a$

DL Assumption: There is no efficient algorithm (polynomial time) to solve DL problem.

- It is widely believed that this assumption holds.

DLP

- When p is small discrete log can be found by exhaustive search

- The size of prime for which DL can be computed is approximately the same as the size of integers that can be factored
  - In 2001: 120 digit DL, 156 digit factorization

- DL is an example of one-way function:
- A function f is **one-way** if
  - Given x, finding f(x) is easy
  - Given y, finding x such that f(x)=y is hard

The El Gamal Cryptosystem

- **Key generation**:
  - Alice chooses a prime **p** and two random numbers **g** and **u**, both less than **p**, where **g is** a generator of $\mathbf{Z}_p^*$.
  - Then she finds:

$$y = g^u \bmod p$$

Alice's public key is **(p, g, y)**, her secret key is **u**.

- To encrypt a message **X** for Alice, Bob chooses a random number **k** such that **gcd(k,p-1)=1**. Then he calculates:

$$a = g^k \bmod p$$

$$b = y^k \times X \bmod p$$

- The cryptogram is **(a,b)**
- The length is twice the length of the plaintext.

- To decrypt **(a,b)** Alice calculates

$$X = \frac{b}{a^u} \bmod p$$

ElGamal Cryptosystem

Public Key : $y = g^u$, *Secret Key* : u

Encryption :

$a = g^k \mod p$

$b = y^k \times X \mod p$, *ciphertext* : (a, b)

Decryption :

$X = \frac{b}{a^u}$, *division uses Euclid extended Algorithm*

To avoid division :

$X = b \times a^{-u} = b \times a^{p-1-u}$

Example

- We choose p=11, g=3, u=6.
- We calculate $y=3^6=3 \bmod 11$

  $3^6 \bmod 11=(3^2)^3 \bmod 11$

  $=(-2)^3 \bmod 11$

  $= -8 \bmod 11$
- The public key is **(11,3)**.

- To encrypt **X=6**, Bob chooses k=7 and calculates:

  $a=3^7=9 \bmod 11, \qquad b=3^7 \times 6=10 \bmod 11$

  The cryptogram is **(9,10)**.

- To decrypt Alice finds:

  $X=10/9^6=10/9=10 \times 5=\mathbf{6} \bmod 11$

Security problems with ElGamal

- Similar to RSA:

$$\textit{Given } (a,b) = (g^k, y^k \times m) \mod p$$

Eve : – *chooses a random number r in* Z_p^*,

– *computes rb*

– *sends (a, rb) to Alice*

Alice returns the decryption: $r \times m \mod p$

Eve will find: $\dfrac{r \times m}{r} = m \mod p$

Provable security

- Adversary power
  - Eve is polynomially bounded
    - time and memory is bounded by a polynomial in the size of input
    - Cannot exhaustively search
  - Eve has access to ‘oracles’:
    - Can ask queries and receive responses
    - Attacks are modeled as a ‘game’ between an attacker Eve and an innocent user (Oracle)

CSCI361
Computer Security

Digital Signatures

Outline

- Digital signatures.
  - Required properties.
- Main differences between handwritten and digital signatures.
- Signature schemes.
  - Component algorithms and requirements on them.
- Digital signatures using PKC.
- A problem and a solution.

Digital Signatures

- Introduced by Diffie-Hellman (1976).
- A digital signature is the electronic analog of handwritten signature. It ensures integrity of the message and authenticity of the sender. That is, if Bob gets a message which supposedly comes from Alice, he is able to check that…
  - The message has not been modified.
  - The sender is truly Alice.

- Digital signatures require a few properties. They should be:
  - Easy to generate.
  - Easy to verify by a receiver.
  - Hard to forge.

- A digital signature is generally represented as a **string of bits** attached to a message.
- This is similar to the mechanism for **M**essage **A**uthentication **C**odes but they didn't allow for the authenticity check on the sender.

Differences between handwritten and digital signatures

- There are a few major differences between handwritten and digital signatures.
  - The digital signature needs to depend on the (*whole*) message in order for the integrity to be verifiable.
    - If you just paste your name at the end of your email message this doesn't prove you sent it!
  - To produce a true signature the signer goes through a physical process. With a digital signature the signer provides input to a process performed by a computer.
    - This input uniquely identifies the signer, it could be a password, or biometric data (fingerprint, retina print, etc) or more likely both.

- There are two major categories of digital signature:
  1. **True digital signatures**: A signed message is generated by the signer and sent directly to the receiver, who can directly verify the signature.
     - A third party is only required in the case of disputes.
  2. **Arbitrated signatures**: In these the signed message is sent via a third party.
- We are only really interested in the first type here.

Signature schemes

- A **signature scheme** consists of two algorithms (possibly three if you consider key generation too):
  1. A **signature generation algorithm** denoted $\mathbf{S_A}$, takes a message **X** and produces a signature $\mathbf{S_A(X)}$.
  2. A **signature verification algorithm** denoted $\mathbf{V_A}$, takes a message **X** and the associated signature and produces a **True** or **False** value.
     - True indicates the message should be accepted while False indicates the message should be rejected.

Algorithm requirements

- The algorithms defining a signature scheme need to satisfy some requirements:
  - **$V_A(X,S)$ = True** if and only if **$S=S_A(X)$**.
  - Without knowledge of the key, it is computationally infeasible to construct a valid pair **(X, S)** such that **$V_A(X,S)$ = True**.
  - To ensure **protection against forgery**, we need the property that **if** $(X,V_A(X,S))$ is known, it is computationally infeasible to find X' such that $V_A(X',S)=V_A(X,S)$.

- In general it is possible to have two messages with the same digital signature. That is, there exists **X** and **X'** such that, $\mathbf{S_A(X)=S_A(X')}$.
- We only require that it is computationally infeasible to find such an **X'** if $\mathbf{(X,V_A(X,S))}$ is known.

Digital Signatures using PKC

- **A**lice signs **X** by creating **Y= E_{z_A} (X)**. The signed message is **(X,Y)**.
- **B**ob validates **A**lice's signature by computing **X'=E_{z_A}(Y)** and comparing it with **X**.

- Everyone has access to **E_{z_A}** and hence anyone can verify the signature by computing **E_{z_A}(Y)**.

- Since **E_{z_A}** is a trapdoor one-way function, it is not possible for an intruder to find **z_A**.
- Hence **A**lice's signature cannot be forged.

Other integrity services

- Digital signatures can provide more than just explicit authentication and integrity.
- For example, they can provide **non-repudiation**. This prevents the sender from denying he/she has sent the message.

- If Alice signs a message, since she is the only one who knows $\mathbf{z_A}$, she will be held accountable for it.

  → The system therefore provides **non-repudiation**.

- **A**lice may be able to repudiate by claiming $\mathbf{E}_{\mathbf{z_A}}$ was compromised.
  - To provide protection against this we may use an *arbitrator or public notary.*
  - A public Notary **N** signs messages on top of **A**lice's signature. Hence **A**lice cannot claim the message is forged. **N** sends a copy of the signed message back to **A**lice.

- **Proof of delivery** is protection against the receiver denying receipt of the message. This is normally implemented by using an **judicable protocol**. These are protocols that can be later verified by a third party.

$$A \rightarrow B : Y = E_{Z_B}(E_{z_A}(X))$$

B computes X', the decryption of the received message.

$$B \rightarrow A : Y' = E_{Z_A}(E_{z_B}(X'))$$

- **B**ob acknowledges receipt of **X**.

- **A**lice assumes the message was not received, unless he receives an acknowledgment from **B**ob.

Some problems …

- As described we would implement a PKC based digital signature by breaking a message into blocks and signing each block independently.

  $X_1, X_2, \ldots, X_t$

  $S_A(X_1), S_A(X_2), \ldots, S_A(X_t)$

- This means, however, the signed message is susceptible to *block re-ordering, block deletion*, and *block replication*.
- It is possible to protect against these by adding redundancy to the blocks. This makes the system less efficient however.

- Problems:
  - PKC is slow. ☹
  - The signature is the same size as the message. ☹

  → It is desirable to produce a **short** signature that is a function of the whole message.

… and a solution

- The solution is to use hash function in conjunction with PKC.
- We will look at hash functions after we have looked at some examples of digital signatures.

CSCI361
Computer Security

Digital Signatures Schemes

Outline

- The Elgamal Signature scheme.
  - Signing and verifying.
  - Example.
- The Digital Signature Standard (DSS).
  - Development history.
  - Digital Signature Algorithm.
- Other types of signature schemes.

The Elgamal signature scheme

- This is based on the difficulty of computing discrete logarithms over $\mathbf{Z_p}$, where **p** is a prime.

- **Setup and Key generation**:
  - **A**lice chooses a large prime **p** such that **p-1** has a large prime factor.
  - Let **g** denote a primitive element of $\mathbf{Z_p}$. **A**lice chooses **x** as her private key and calculates $\mathbf{y=g^x \bmod p}$.

- The public key is **(p, g, y)**.
- The private key is **x.**
- This is the same as for encryption.

Signing and verifying

- To sign a message X:
  - **A**lice chooses and integer *k, 1≤ k ≤ p-1,* such that **gcd(k,p-1)=1**.
  - **A**lice calculates
    - **$r=g^k \bmod p$**.
    - **$s=k^{-1}(X-xr) \bmod (p-1)$**
  - The signed message is **(X.(r,s))**.

Using the received values

- To verify the message; compare g^x and $y^r r^s$.

$$g^X = g^{xr+ks} = g^{xr} * g^{ks} = y^r * r^s$$

- **Example**: p=11, g=2. (Remember in previous notes we showed that 2 is a generator of $\mathbf{Z_{11}}$.

- **Private key**: x=3.
- **Public key**: $y=2^3=8 \bmod 11$.

- **Signing X=9**.
  - Choose **k=7**. ➔ Check that **gcd(7,10)=1**. ✓
  - Calculate $\mathbf{k^{-1}=3 \bmod 10}$. ➔ Calculated with above!
  - Calculate $\mathbf{r=g^k=2^7=7 \bmod 11}$.
  - Calculate **s=3(9-7*3) mod 10 = 4**.
- The signed message is **(9.(7,4))**.

- **Verification** is by comparison of g^x and $y^r r^s$.
  $2^9= 6= 8^{7*}7^4 \bmod 11$

The Digital Signature Standard (DSS)

- Proposed by NIST in 1991, finally issued in 1994 (Federal Information Processing Standard FIPS 186).
- Various modifications were made, FIPS 186-2 came out in 2000.
- Work on FIPS 186-3 is currently underway (there is a draft).
- We are just going to talk about the original Digital Signature Algorithm in FIPS 186.

Digital Signature Algorithm (DSA)

- **Key generation and setup**:
  - **q** a 160-bit prime is chosen
  - **p** a prime. 512-1024 bits long with the length a multiple of 64. **q** is a factor of **p-1**.
  - **h** a number less than **p-1** such that

    $$g = h^{p-1/q} \bmod p > 1$$

  - **u < q**
  - **$y=g^u \bmod p$**

- **Public parameters**: p, q, g. (Multiple people could use these).
- **Private key**: u.
- **Public key**: y.

Generating and verifying a signature

- To sign a message …
  - **A**lice generates a random **k**, such that **k<q**.
  - **A**lice computes

    $r = (g^k \bmod p) \bmod q$

    $s = (k^{-1}(H(X) + ur))) \bmod q$
- To verify a message Bob calculates…

  $w=s^{-1} \bmod q$

  $t_1=(H(X)*w) \bmod q$

  $t_2=rw \bmod q$

  $v=((g^{t_1}*y^{t_2}) \bmod p) \bmod q$

  … and accepts the signature if **v=r**.

Using the received values

H(X) is the hash of X. We will get to these soon.

Example

- **Setup**:
  - p=29, q=7 (is a factor of 28).
  - h=3, $g=h^{(29-1)/7}=3^4=23 \bmod 29$
  - u=2 < q.
  - $y=g^2 \bmod 29 = 23^2 \bmod 29$

    $= (-6)^2 \bmod 29$

    $= 36 \bmod 29 = 7$

- **Signing:**
  - To sign **H(X)=5**, **A**lice chooses **k=4** (<q) and calculates $k^{-1}=2$.

    $r = (g^4 \bmod 29) \bmod 7$

    $= ((-6)^4) \bmod 29) \bmod 7$

    $= (36*36 \bmod 29) \bmod 7$

    $= (7*7 \bmod 29) \bmod 7$

    $= (49 \bmod 29) \bmod 7 = 20 \bmod 7 =$ **6**

    $s = k^{-1}(H(X) + ur) \bmod q$

    $= 2(5+2*6) \bmod 7$

    $= 34 \bmod 7$

    $=$ **6**

- **Verifying**:

  $w = s^{-1} \bmod q = 6^{-1} \bmod 7 = \mathbf{6}$

  $t_1 = (H(X)*w) \bmod q$

  $= 5*6 \bmod 7 = \mathbf{2}$

  $t_2 = rw \bmod q$

  $= 6*6 \bmod 7 = \mathbf{1}$

  $v = ((g^{t_1}*y^{t_2}) \bmod p\ ) \bmod q$

  $= ((23^2*7^1) \bmod 29) \bmod 7$

  $= (((-6)^2*7) \bmod 29) \bmod 7$

  $= ((7*7) \bmod 29) \bmod 7$

  $= \mathbf{6}$

- Since **v=r** the message and signature are accepted as authentic.

Notes

- The recommended hash function for DSA is **SHA-1**.
  - We will discuss this later.

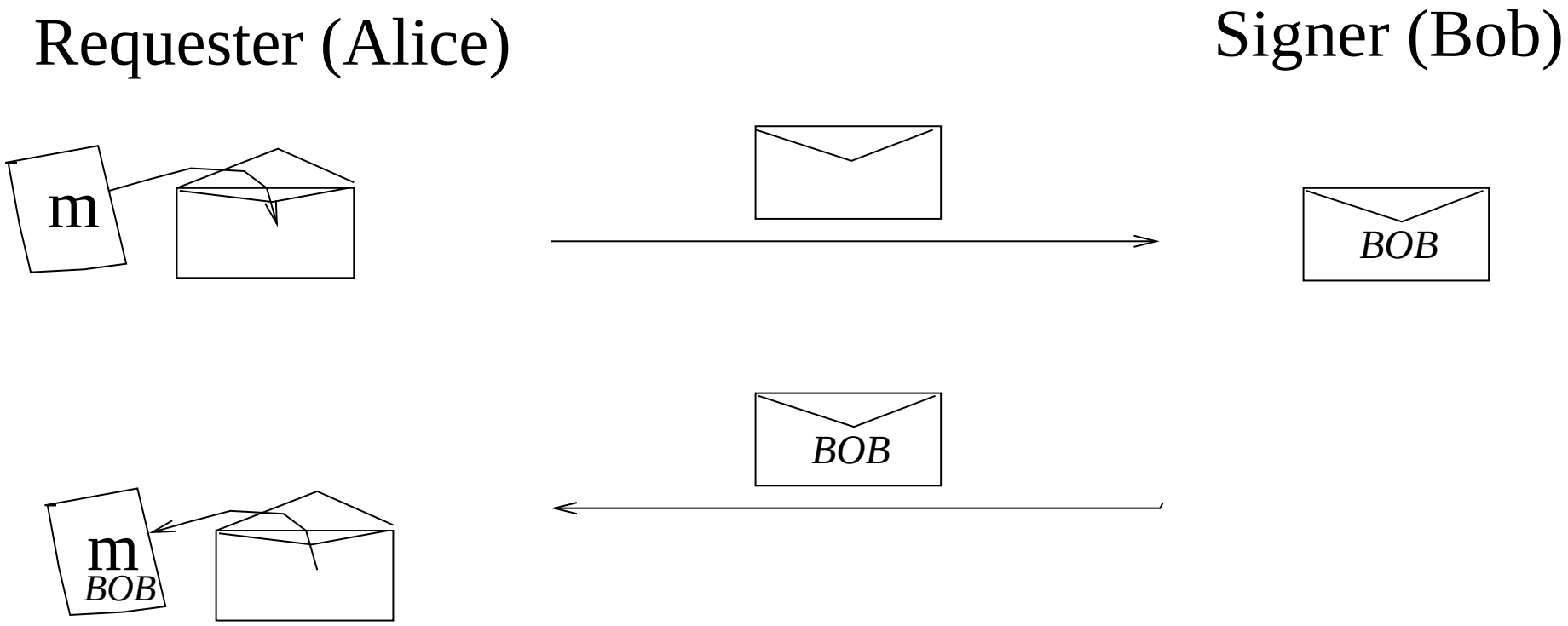
- There were several criticisms against DSA.
  - DSA cannot be used for encryption or key distribution.
  - It was developed by NSA and so cannot be trusted.
  - RSA is the de facto standard.
  - The key size is too small.

Signature scheme variants

- There are many different types of signature schemes.
- They are designed to meet the requirements of different scenarios.
- The cost of adding properties is efficiency, both in terms of computation and storage.
  - A lot of research is done into improving the efficiency of various different types of signature schemes.
- We are going to look briefly at a few types of signature schemes.

Blind signatures

- In some cases it is necessary to get the signature of a party without allowing them to see the message. For example, in electronic cash (proposed by David Chaum), a coin is the means of payment. A coin has a serial number and must be signed by the bank to be authentic.
  - To generate a coin, a customer randomly generates a number. The customer needs the bank signature on the coin before being able to spend it. To ensure privacy of the coin, that is to ensure that the bank cannot link a coin with a particular customer, the customer uses a **blind signature scheme**.
- In general:
  - The requester wants to obtain the signer's signature of message *m*.
  - The requester doesn't want to reveal *m* to anyone, including the signer.
  - The signer signs *m* blindly, not knowing what they are signing.
  - The requester can retrieve the signature.


Requester (Alice)
Signer (Bob)
m
BOB
BOB
m
BOB

Blind signatures using RSA

- **Setup**: **B**ob has (d,e), a (private,public) key pair.
  - *N*=*pq*, where *p, q* are large primes, associated with **B**ob.
- For a message **X**, that **A**lice wants **B**ob to sign, **A**lice constructs $\mu = mr^e \bmod N$, where $\mathbf{r} \in_R Z_N^*$, and sends μ to **B**ob.
- **B**ob signs μ and sends his signature $\sigma = \mu^d \bmod N$ back to **A**lice.
- **A**lice retrieves the signature *s* of *m* by computing:

$$s = \frac{\sigma}{r} = \frac{\mu^d}{r} = \frac{m^d r^{ed}}{r} = \frac{m^d r}{r} = m^d \bmod N$$

Undeniable signatures

- These are non-transferable signatures.
- That is, the signature can be verified only with the collaboration of the signer.
  - This could be useful, for example, when a software distributor signs software. In this case it is important that the verifiable software cannot be duplicated. That is, the signature can be different for different people and the distributor will be able to see if there is duplication.

Group signatures

- Consider the following scenario:
  - A company has several computers connected to the local network.
  - There are a number of departments, each having a printer.
  - The printer for each department is only allowed to be used by members of that department.
  - So before printing, the printer must be convinced that the user is a member of that department.
  - To ensure privacy the user's name may not be revealed.
  - However if a printer is used too often, the director must be able to discover who misused the printer.
- Group signatures (with revocation) allow us to met these requirements.

- A **group signature scheme** has the following properties:
  - Only members of the group can sign messages.
  - The receiver can verify the signature but *cannot discover which member of the group generated it*.
  - In the case of a dispute, the signature can be “opened” to reveal the identity of the signer.

Fail-stop signature schemes

- A Fail-Stop Signature Scheme provides enhanced security against the possibility that a very powerful adversary might be able to forge a signature.

- In the event that Oscar is able to forge Bob's signature on a message, Bob will (with high probability) subsequently be able to *prove* that Oscar's signature is a forgery.

Threshold signature schemes

- Consider the following scenario:
  - In a bank, there is a vault which must be opened everyday.
  - The bank employs three senior tellers, but they do not trust the combination to any individual teller.
  - We would like to design a system whereby any two of the three senior tellers can gain access to the vault, but no individual teller can do so.
- In the context of a signature basically the parties need to work together to construct a **proof of authority**.
- This problem can be solved by means of a *secret sharing scheme*, we will be talking about those later in this course.

- A **(t, w)** threshold scheme is a method of sharing a key **K** among a set of **w** participants in such a way that any **t** participants can compute the value of **K**, but no group of **t-1** participants can do so.