Computer and Network Security

Computer & Information Sciences & Engineering
University Of Florida
Gainesville, Florida 32611-6120
nemo@cise.ufl.edu

Public Key Cryptography
(Pfleeger Ch. 3, KPS Ch. 5, 6)

Group Theory and Intractibility

1 Intro to PKC

1.1 Role of PKC

Help for symmetric cryptography...

1. Traditional cryptography is symmetric: both principals hold a common secret (the key) used for both encryption and decryption.
2. There, the primary problem is key distribution.
3. PKC can help solve that problem by providing cryptographically secure channels for key distribution.

Other properties

1. Message signing that can be verified by anyone with non-repudiation
2. New problem: secure association of the key with the principal
3. Leads to some very deep philosophical issues
4. Solutions lead to such concepts as the "web of trust" and PKI
5. PKC is usually very slow, because it relies on large integer arithmetic
6. Typically used to send symmetric key that is used for encrypting and decrypting messages
7. To sign a message, rather than signing the whole message, a small message digest is computed using one of the one-way hash functions and that result is then signed.

2 Basics of PKC

1. Two keys are used instead of one.
2. One key made public, so secure channel not necessary
3. Other key kept secret, available only to principal with which key pair associated.
4. If e is the public key and d is the private key, then for plaintext p, ciphertext c is

   $$c = E(p, e)$$

   which is decrypted by

   $$p = D(c, d) = D(E(p, e), d)$$

5. Anyone with e can encrypt a message, but only keyholder with d can decrypt it.

6. To sign p, use reverse operations:

$$s = D(p, d)$$

to verify, recipient checks that

$$v = E(s, e) = E(D(p, d), e) = p$$

7. For this, necessary that encryption and decryption commute, but they do in most cases.
8. If not, then different signing key used and the message is encrypted with it, then verified by decrypting with public verification key.

3 Modular Arithmetic

3.1 Groups

3.1.1 Definition

A group is a pair $G = \langle S, \diamond \rangle$, where S is a set and $\diamond$ is a binary function on S, that satisfies the four following requirements (CAIN).

1. (C) Closed:
$$\forall\ a, b \in S,\ a \diamond b \in S$$
2. (A) Associative:
$$\forall\ a, b, c \in S,\ (a \diamond b) \diamond c = a \diamond (b \diamond c)$$
3. (I) Identity:
$$\exists!\ e \in S \text{ such that } \forall\ a \in S,\ a \diamond e = e \diamond a = a$$
4. (N) Inverse:
$$\forall\ a \in S,\ \exists!\ a^{-1} \in S \text{ such that } a \diamond a^{-1} = a^{-1} \diamond a = e$$

3.1.2 Types of Groups

Groups may be

1. finite or infinite (we are interested in finite groups)
2. commutative or not

- $G = \langle S, \diamond \rangle$ is commutative iff

$$\forall\, a, b \in S,\ a \diamond b = b \diamond a$$

3.1.3 Examples of Groups

1. $\langle \text{Integers}, + \rangle$
2. $\langle \text{Rationals-}\{0\}, \times \rangle$
3. $\langle \text{Polynomials}, + \rangle$
4. $\langle [0..N-1], + \; modulo \; N \rangle$
5. $\langle [1..P-1], \times \; modulo \; P \rangle$ where P is a prime number
6. $\langle \{x \in [1..N] : (x, N) = 1\}, \times \; modulo \; N \rangle$
7. $\langle \Pi(S), \circ \rangle$, where $Pi(S)$ is the set of permutations on some set S and $\circ$ is function composition

Note $(x, N) = GCD(x, N)$

3.2 Euler's Totient Function

Define the set of positive integers less than N and relatively prime to N to be

$$\Phi(N) = \{x \in [1..N] : (x, N) = 1\}$$

Euler's Totient Function is defined as

$$\phi(N) = |\Phi(N)|$$

It is the number of positive integers less than N that are relatively prime to N.

If N is prime, then

$$\Phi(N) = [1..N-1]$$

and

$$\phi(N) = N - 1$$

If $N = pq$, where p and q are prime, then

$$\phi(N) = (p-1)(q-1)$$

3.2.1 Modular Exponentiation

Fact: If N is square-free (i.e., it does not have p^2 as a factor for any prime p), then

$$\forall x \in [1..N], x^y \ modulo\ N = x^{y \ modulo\ \phi(N)} \ modulo\ N$$

If

$$y = 1 \ modulo\ \phi(N),$$

then

$$x^y = x^1 = x \ mod\ N$$

4 RSA

4.1 RSA Algorithm

1. Pick large primes p and q, let $N = pq$
2. Pick $e \in [1..N-1]$, $(e, \phi(N)) = 1$
3. Public Key $= \langle e, N \rangle$
4. Find d, the multiplicative inverse of e *modulo* $\phi(N)$.
5. Private Key $= \langle d, N \rangle$

To encrypt message $m < N$, (if larger, then break into chunks), compute

$$c = m^e \; modulo \; N$$

To decrypt, compute

$$m = c^d \; modulo \; N = (m^e \; modulo \; N)^d \; modulo \; N$$
$$= m^{de} \; modulo \; N = m \; modulo \; N = m$$

To sign, compute

$$s = m^d \; modulo \; N$$

To verify a signature, compute

$$\begin{aligned} v = s^e \; modulo \; N = (m^d \; modulo \; N)^e \; modulo \; N \\ = m^{de} \; modulo \; N = m \end{aligned}$$

and compare v to m

4.2 Security of RSA

Factoring large numbers appears to be very hard, and that is the only way we theoretically know of to break RSA *as an algorithm*.

4.3 Implementation of RSA

4.3.1 Exponentiation of Large Integers

Perform exponentiation using the binary expansion of the exponent:

$$x^a = a_{n-1}x^{2^{n-1}} a_{n-2}x^{2^{n-2}} ... a_1 x^1 a_0 x^0$$

where

$$a = a_{n-1}a_{n-2}...a_1a_0$$

in binary.

Precompute the powers of x to powers of 2 by repeated squaring

$$x, x^2, x^4, x^8, ...$$

Perform modular reduction after each multiply

Total work is linear in the number of bits of a.

4.3.2 Finding large primes

Use Euler's Theorem to guess and verify

4.4 Attacks on RSA

1. Factoring
2. Limited plaintext attack
3. System approach

5 PKCS - Public Key Cryptography Standards

RSADSI has produced a number of standards for use of PK crypto so that the subtle ways in which the system may be used in an insecure way are avoided, and so that different organizations can build software that interoperates.

See http://www.rsasecurity.com/standards/

1. PKCS #1:RSA Cryptography Standard
2. PKCS #2: incorporated into PKCS #1
3. PKCS #3:Diffie-Hellman Key Agreement Standard
4. PKCS #4: incorporated into PKCS #1
5. PKCS #5:Password-Based Cryptography Standard
6. PKCS #6:Extended-Certificate Syntax Standard
7. PKCS #7:Cryptographic Message Syntax Standard
8. PKCS #8:Private-Key Information Syntax Standard
9. PKCS #9:Selected Attribute Types
10. PKCS #10:Certification Request Syntax Standard
11. PKCS #11:Cryptographic Token Interface Standard
12. PKCS #12:Personal Information Exchange Syntax Standard
13. PKCS #13: Elliptic Curve Cryptography Standard
14. ???
15. PKCS #15: Cryptographic Token Information Format Standard

5.1 PKCS#1

5.1.1 RSA encryption

- Format:

$$[0|2| \geq 8 \text{ random bytes}|0|\text{data}]$$

- The MSByte of zero is to guarantee that the message is less than the modulus
- The 2 is a format type (1 is for signatures)
- The random bytes are non-zero so that zero can be used to delimit the data from the padding, and for confusion (this so-called confounder will prevent the same data from looking the same after encryption).
- The 0 delimits the pad from the data.

- The confounder helps eliminate the threat of encrypting a limited number of possible messages and comparing to the encrypted message sent - the attacker would have to guess the padding as well.
- It also prevents problems from sending the same message to multiple recipients when e=3 as long as the padding is different for each recipient.
- If e=3, then it prevents the problem of sending a message that is no more than one third the length of the modulus - the second byte is non-zero so this will never happen (the cubic root attack).

5.1.2 RSA Signing

- Format:

$$[0|1| \geq 8 \text{ ff bytes}|0|\text{formatted digest}]$$

- the formatted digest is the digest type and the digest encoded in ASN.1
- The MSByte of zero is to guarantee that the message is less than the modulus
- The 1 is the format type for signatures
- The padding bytes ensure that the number is large and unlikely to be a smooth number
- The 0 delimits the pad from the digest

- The digest formatting includes the digest type to prevent attacks that use a weak digest type to produce a different message with the same digest as the digest you used, and then have the recipient believe that you signed the other message with the weaker digest type.

6 Diffie-Hellman

6.1 Comments

- Oldest PKC still in use
- Provides a means for generating a shared secret
- Will not provide authentication by itself
- A and B start out with no knowledge at all
- They exchange some information publically
- Afterwards, each will be able to derive a secret that nobody else knows, even though the exchange was public.
- Once A and B have a secret, they can use this to derive symmetric keys for conventional encryption, signing, etc.

6.2 Diffie-Hellman Protocol

1. A and B publically announce prime number p and base $g < p$
2. A picks random S_A,
   B picks random S_B
3. A computes $T_A = g^{S_A}$ *modulo* p,
   B computes $T_B = g^{S_B}$ *modulo* p
4. A sends T_A to B, B sends T_B to A
5. A computes $K = T_B^{S_A}$ *modulo* p,
   B computes $K = T_A^{S_B}$ *modulo* p
6. Shared secret K is

$$
\begin{aligned}
K \;&=\; T_B^{S_A} \textit{ modulo } p \;=\; (g^{S_B} \textit{ modulo } p)^{S_A} \textit{ modulo } p \\
&=\; g^{(S_B)(S_A)} \textit{ modulo } p \;=\; g^{(S_A)(S_B)} \textit{ modulo } p \\
&=\; (g^{S_A} \textit{ modulo } p)^{S_B} \textit{ modulo } p \;=\; T_A^{S_B} \textit{ modulo } p
\end{aligned}
$$

- A and B now share secret K, which nobody without knowledge of S_A or S_B can derive.
- To derive S_A or S_B, an attacker must compute the *discrete logarithm* of a large number, which is believed to be very hard.
- S_A = discrete log base g of T_A *modulo* p

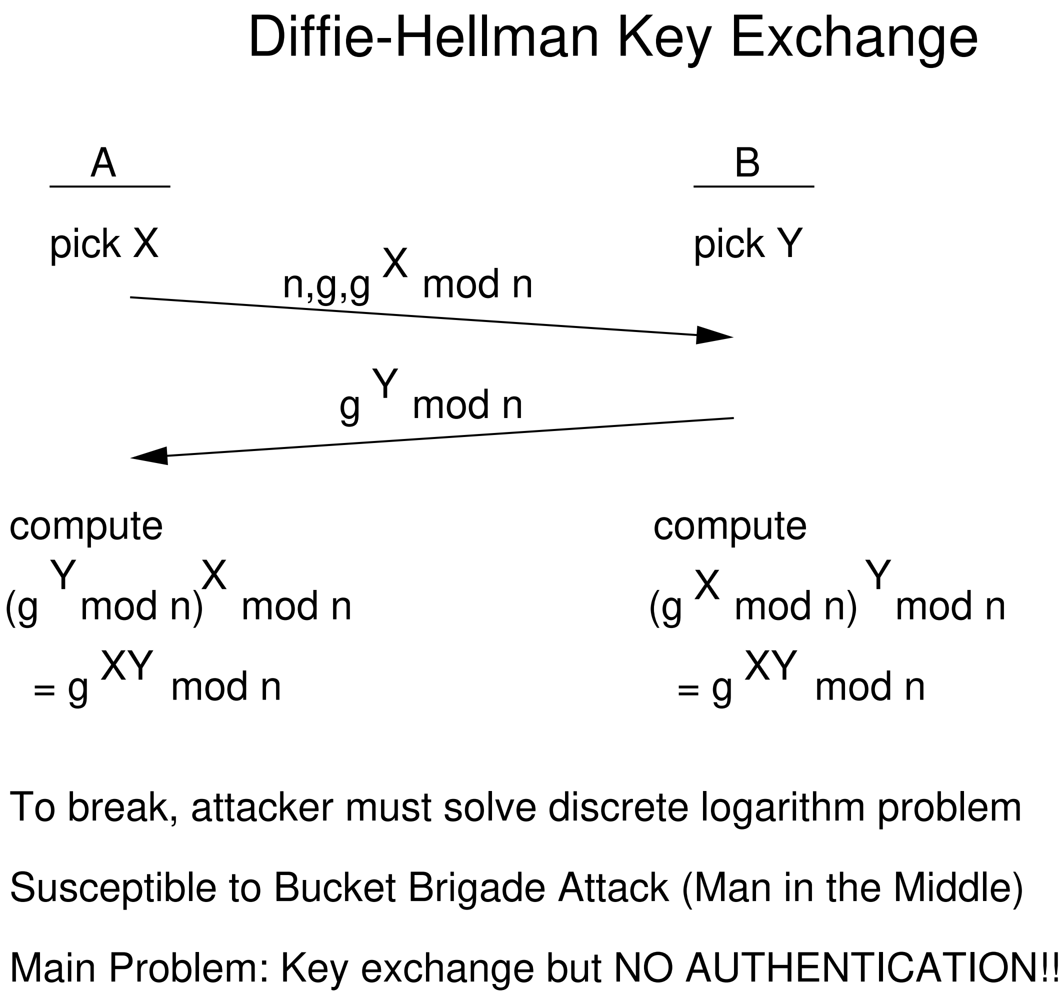

Figure 1: Diffie-Hellman Public Key Exchange

6.3 Bucket-Brigade Attack on Diffie-Hellman

If C interposes herself between A and B, then C can spoof A and B into believing that she is the other party.

To combat this, the p, g, and T may be made public for each principal and distributed reliably.

Alternatively, all members of a group may agree upon a common p and g, then each member A is associated with T_A. This is done in SSL.

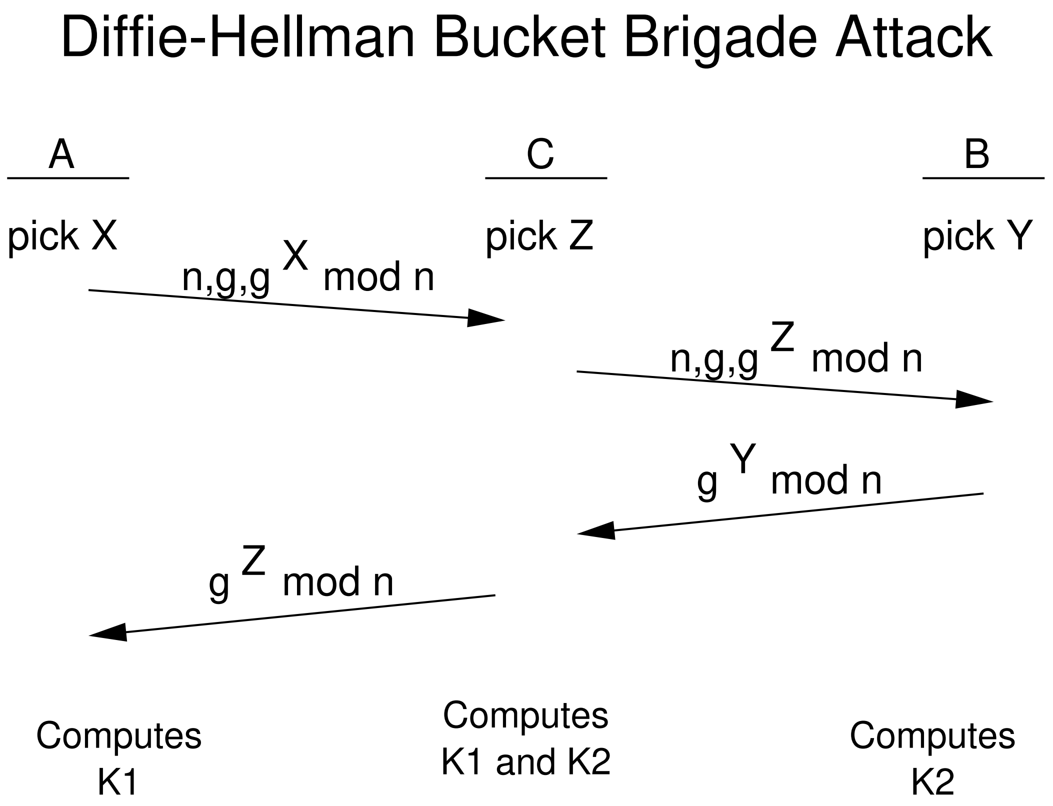

Figure 2: Bucket Brigade Attack on Diffie-Hellman Public Key Exchange

6.4 Encryption with Diffie-Hellman

If the first scheme above is used, then B may publish $\langle p_B, g_B, T_B \rangle$ for anyone to access.

A may then pick S_{AB} and compute

$$T_{AB} = g_B^{S_{AB}} \; modulo \; p_B$$

as well as

$$K_{AB} = T_B^{S_{AB}} \; modulo \; p_B,$$

then send B the message encrypted with K_{AB} along with T_{AB}, which B then uses to generate K_{AB} and decrypt the message.

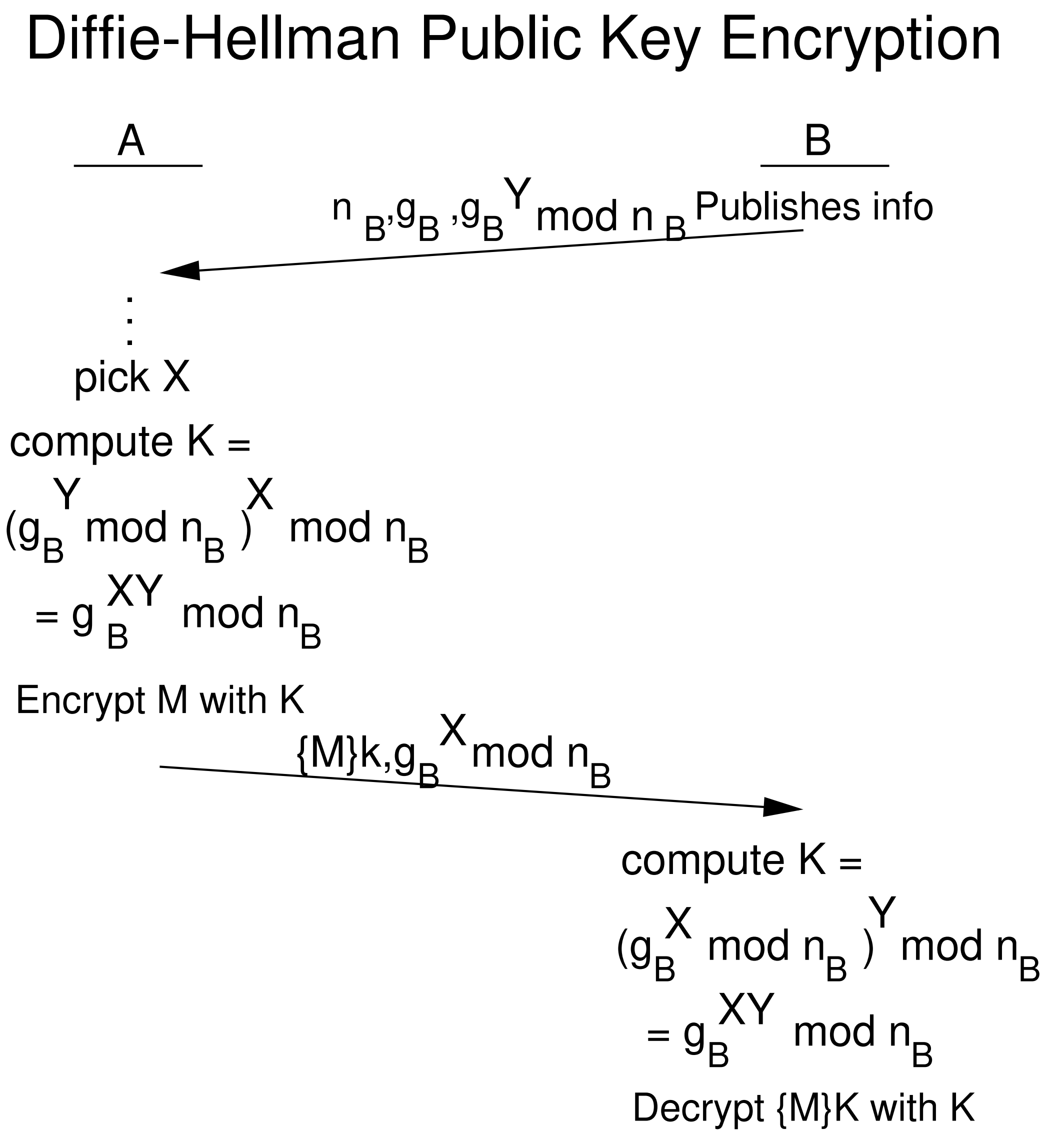

Figure 3: Diffie-Hellman Public Key Encryption

7 El-Gamal Signatures

7.1 El-Gamal Algorithm

- Each principal has long-term, public/private key pair

$$(\langle g, p, T \rangle, S),$$

- where the public $\langle g, p, T \rangle$ and the private S are as in D-H:

$$T = g^S \; modulo \; p.$$

To sign a message M,

- the signer generates a new secret, S_M
- computes
$$T_M = g^{S_M} \; modulo \; p$$
- A digest d_M is then computed of M concatenated with T_M,
$$d_M = H(M|T_M).$$
- Then a signature X is computed by
$$X = S_M + d_M S \; modulo \; (p-1).$$
- The message M is sent along with X and T_M.

To verify, on receipt of M, X and T_M, the recipient

- computes d_M (knowing H, M and T_M),
- then checks that

$$g^X = T_M T^{d_M} \ \mathit{modulo}\ p,$$

since

$$\begin{aligned} g^X \ \mathit{modulo}\ p &= g^{S_M + d_M S \ \mathit{modulo}\ (p-1)} \ \mathit{modulo}\ p \\ &= g^{S_M} g^{d_M S} \ \mathit{modulo}\ p \\ &= (g^{S_M} \ \mathit{modulo}\ p)(g^{d_M S} \ \mathit{modulo}\ p) \ \mathit{modulo}\ p \\ &= T_M (g^S)^{d_M} \ \mathit{modulo}\ p \\ &= T_M T^{d_M} \ \mathit{modulo}\ p \end{aligned}$$

All of these are available, since the recipient knows g, p, T, T_M, M, X and H, and can compute d_M and the other quantities from these.

8 DSS - Digital Signature Standard (NIST 1991)

8.1 Comments

- uses DSA - Digital Signature Algorithm
- based on El-Gamal, which uses primes of 512 bits or more,
- modified to used a modulus of q, where q is a prime of 160 bits that divides $p - 1$.

8.2 DSS Algorithm

1. Generate p and q as above: find 160-bit prime q, then find a 512-bit prime p of the form $kq + 1$.
2. Generate g, where $g^q = 1$ mod p, by generating random number h and taking
$$g = h^{(p-1)/q} \; modulo \; p,$$
since
$$g^q = (h^{(p-1)/q})^q = h^{p-1} = 1 \; mod \; p \quad \text{(Fermat)}$$
(Note: if this $g = 1$, try again!)
3. Choose long-term public/private key pair $\langle T, S \rangle$ by chosing random $S < q$ and computing $T = g^S \; modulo \; p$
4. Make p, q, g, and T public.

To sign a message M,

1. On a per-message basis, choose $\langle T_M, S_M \rangle$ by choosing a random S_M and computing
$$T_M = (g^{S_M} \; modulo \; p) \; modulo \; q$$
2. Compute $S_M^{-1} \; mod \; q$
3. Calculate $d_M = H(M)$ for some hash function H.
4. Compute the signature $X = S_M^{-1}(d_M + ST_M) \; modulo \; q$
5. Send M, T_M and X.

To verify, on receipt of M, T_M and X,

1. Calculate X^{-1} $mod\ q$ and d_M, then
2. $x = d_M X^{-1}\ mod\ q$
3. $y = T_M X^{-1}\ mod\ q$
4. $z = (g^x T^y\ mod\ p)\ mod\ q$
5. and check that $z = T_M$.

If we define

$$v = (d_M + ST_M)^{-1} \; mod \; q,$$

then

$$\begin{aligned} X^{-1} &= (S_M^{-1}(d_M + ST_M))^{-1} \; mod \; q \\ &= (d_M + ST_M)^{-1}(S_M^{-1})^{-1} \; mod \; q \\ &= (d_M + ST_M)^{-1} S_M \; modulo \; q \\ &= S_M(d_M + ST_M)^{-1} \; modulo \; q \\ &= S_M v \; modulo \; q \end{aligned}$$

and

$$x = d_M X^{-1} \; modulo \; q = d_M S_M v \; modulo \; q$$

$$y = T_M X^{-1} \; modulo \; q = T_M S_M v \; modulo \; q$$

so

$$z = (g^x T^y \; modulo \; p) \; modulo \; q$$

$$= (g^{d_M S_M v \; modulo \; q} T^{T_M S_M v \; modulo \; q}) \; modulo \; p) \; modulo \; q$$

$$= (g^{d_M S_M v} g^{ST_M S_M v} \; modulo \; p) \; modulo \; q$$

$$= (g^{(d_M + ST_M) S_M v} \; modulo \; p) \; modulo \; q$$

$$= (g^{(d_M + ST_M) S_M (d_M + ST_M)^{-1}}) \; modulo \; p) \; modulo \; q$$

$$= (g^{S_M} \; modulo \; p) \; modulo \; q = T_M$$

Note: the *mod q* in the exponents of g is OK since

$$g^q = 1 \; mod \; p$$

9 Attacks on El-Gamal and DSS

- Signer must generate a unique secret number for each new message
- If same secret number were used for two messages, then the signer's long-term secret number would be exposed
- If that is the case, then anyone may forge the signature

Let

- messages M and M' be DSS signed using the same secret number S'
- X and X' be the signatures
- d and d' be the hashes of M and M', respectively

Then

$$(X - X')^{-1}(d - d') \; modulo \; q = S' \; modulo \; q.$$

Given a per-message secret S' for message with signature X using T' and having hash d, compute

$$(XS' - d)T'^{-1} \; modulo \; q = S \; modulo \; q.$$

That is sufficient to generate signatures.

10 Zero-Knowledge Proof Systems

10.1 General Comments on ZKPSs

A zero-knowledge proof system does only authentication by proving that you know a secret associated with your public key without revealing that secret.

RSA does that by allowing you to prove you know the private key without revealing it.

A class of these are based on NP-hard problems, such as graph isomorphism.

10.2 Graph Isomorphism Example of ZKPS

1. A generates a large graph G
2. A renames the vertices of G to produce G', preserving the mapping between G and G'
3. These two graphs (G, G') are A's public information.

When B wants to authenticate A,

1. B asks A for a set of graphs $\{G_1, G_2, ..., G_k\}$ isomorphic to G (and G')
2. A generates these and sends them to B, preserving the mapping from G to each G_i
3. B then may, for each G_i, request the mapping from either G or G' to G_i, but not both
4. B can easily verify the mappings to authenticate A

- If a third party C tries to impersonate A, then there would only be a 50-50 chance that B requests the same mapping for the same graph, even if the same graphs had been used before,
- so the probability the C gets lucky on all k graphs is

$$(1/2)^k,$$

which may be made arbitrarily small by making k large.
- As long as A never reveals a mapping for any graph to both G and G', the mapping from G to G' will remain secret.

10.3 An Almost ZKP Authentication Scheme

A more efficient, almost-zero knowledge authentication scheme works as follows.

1. A picks large primes p and q, and computes a modulus n,
$$n = pq$$
as in RSA
2. A picks a random number x and squares it modulo n to produce
$$v = x^2 \; modulo \; n$$
v is a number for which only A knows the square root mod n
3. A publishes $\langle n, v \rangle$,

When B wants to authenticate A,

1. A chooses k random numbers $r_1, r_2, ..., r_k$,
2. for r_i A sends $s_i = r_i^2$ *modulo* n to B
3. B partitions the s_i's into two sets, S and T, then sends these to A
4. For each $s_i \in S$, A sends xr_i *modulo* n to B
5. For each $s_i \in T$, A sends r_i to B
6. For each $si \in S$, B checks that the reply is vs_i *modulo* n,
7. For each $s_i \in T$, B checks that the reply is r_i (i.e., $s_i = r_i^2$ *modulo* n)

If C tries to impersonate A,

1. C may have some s_i's from previous exchanges for which C knows the response for set S
2. C may also generate some numbers and square them to get some numbers for which C knows the square root mod n
3. But B partitions these two types of numbers into the sets S and T
4. So for each number there is only a 50-50 chance that C will have the correct answer available
   C will only know xr_i for the s_i from the previous exchanges, but not r_i;
   C will know r_i for the numbers C generates, but not xr_i)
5. With k numbers, the probability that C gets away with it is $(1/2)^k$, which may be made arbitrarily small.

10.4 Signing with ZKPSs

1. In the authentication scheme, there is an interactive exchange, where B queries A in an unpredictable manner about the data A has sent B
2. In a signature scheme, there is no interaction
3. So a one-way hash takes its place
4. Hash is not predictable from the message, so bits of the hash are taken to represent the queries
5. Since the data items on which the queries are based are included in the message, it is not possible to rearrange them so as to answer only the questions the signer may want to answer
6. The number of questions may be higher to reflect the opportunity that the signer has to try many different message/data combinations in order to find one that only asks for the information he has