Plan of Talk

- ElGamal Digital Signature
- Digital Signature Standard

Plan of Talk

- In RSA, signature and encryption functions are complementary operations.
- Can we find similar feature for ElGamal encryption?
- How to create a signature algorithm based on the hardness of discrete logarithms and Computational DH problems?

Requirements of Digital Signature

- User should be able to define his secret and its public key should be protected by a one way function (some hardness assumption).
- For any message, the user should be able to create a digital tag efficiently using its private secret.
- Other users who possess the user's public key should be able to verify that the signature was indeed created by the owner of the secret.
- The signature should have non-repudiation property (the signer cannot deny creating the signature).
- Signature should be unforgeable.

Some details from theory of numbers

- Consider q a prime number and a generator "a".
- There are q-1 integers less q which are relatively prime to q.
- The generator "a" is chosen such that
- $a^{(q-1)} = 1 \bmod q$
- a is a generator of the group under multiplication.
- Because of the above property we have:
- $a^{(q-1)} = 1 \bmod q$
- Also,
- $a^{t(q-1)} = 1 \bmod q$, for any integer "t".
- In fact,
- $a^{m} = 1 \bmod q$, for any integer satisfying $m = 0 \bmod (q-1)$
- We can have a stronger result:
- $a^{m} = 1 \bmod q$, if and only if $m = 0 \bmod (q-1)$

More details

- Now consider any integer “I” in the range $1 <= i < q-1$.
- Consider q (a prime) and a generator “a” as before:$a^{(q-1)} = 1 \bmod q$
- Because of the above property we have:
- $a^{(i+(q-1))} = a^{(i)} \bmod q$
- In fact adding any multiple of (q-1) to the exponent does not change the result:
- $a^{(i+t(q-1))} = a^{(i)} \bmod q$ for any integer “t”.

- We can have a stronger result:
- $a^{i} = a^{j} \bmod q$, if and only if $i = j \bmod (q-1)$

Direct Digital Signature

- Only source and destination is involved in defining “direct digital signature”.
- Assumption is that the destination knows the public key of the source.
- The validity of the scheme depends on the security of the private key.

How does ElGamal Signature work?

- As before, each user (eg. Alice) generates their key
  - Chooses a secret key (number): $1 < x_A < q-1$
  - Compute their **public key**: $y_A = a^{x_A} \bmod q$
- We would like signature generation depends on the secret x_A and possibly some new secret.
- The signature should depend on the message.
  - Works on exponents (modulo q-1)
- It should be verifiable using only public parameters.
  - Works on finite field (modulo q)

How does ElGamal Signature work?

- Idea: If $x' = (x1 + x2) \bmod (q-1)$, (a^{x1}) and (a^{x2}) are given by a user, without revealing x1 and x2.

- then this can be verified by

- $a^{x'} = (a^{x1})(a^{x2}) \bmod (q)$

- Only the person how knows x1 and x2 could have constructed this sequence: x' , (a^{x1}) and (a^{x2}).

ElGamal Signature Idea

- Given Alice's public key: $y_A = a^{x_A} \bmod q$, and
- Alice private key: $1 < x_A < q-1$ and a message M
- m = H(M) = Hash of the message M
- m = Function(x_A , y_A , S1, S2)
- When LHS ad RHS are applied as exponents of x, the verification should involve only public parameters.
  - $S_1 = a^k \bmod q$
  - $k\ S_2 + x_A\ S_1 = m \bmod (q-1)$
  - Take a^{th} power of LHS and RHS
  - $(S_1^{S2})(y_A^{S1}) = (a^m) \bmod q$
  - Note that verification involves only public parameters.

ElGamal Digital Signatures

- Signature variant of ElGamal, related to D-H
  - so uses exponentiation in a finite (Galois) field
  - with security based difficulty of computing discrete logarithms, as in D-H
- use private key for encryption (signing)
- uses public key for decryption (verification)
- each user (eg. A) generates their key
  - chooses a secret key (number): $1 < x_A < q-1$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$

ElGamal Digital Signature

- Alice signs a message M to Bob by computing
  - the hash $m = H(M)$, $0 <= m <= (q-1)$
  - chose random integer K with $1 <= k <= (q-1)$ and $gcd(K,q-1)=1$
  - compute temporary key: $S_1 = a^k \bmod q$
  - compute k^{-1} the inverse of k $\bmod (q-1)$
  - compute the value: $S_2 = k^{-1}(m-x_A S_1) \bmod (q-1)$
  - signature is: (S_1, S_2)
- any user B can verify the signature by computing
  - $V_1 = a^m \bmod q$
  - $V_2 = y_A^{S_1} S_1^{S_2} \bmod q$
  - signature is valid if $V_1 = V_2$

ElGamal Signature Example

- use field GF(19) $q=19$ and $a=10$
- Alice computes her key:
  - A chooses $x_A=16$ & computes $y_A=10^{16} \bmod 19 = 4$
- Alice signs message with hash $m=14$ as $(3,4)$:
  - choosing random $K=5$ which has $\gcd(18,5)=1$
  - computing $S_1 = 10^5 \bmod 19 = 3$
  - finding $K^{-1} \bmod (q-1) = 5^{-1} \bmod 18 = 11$
  - computing $S_2 = 11(14-16.3) \bmod 18 = 4$
- any user B can verify the signature by computing
  - $V_1 = 10^{14} \bmod 19 = 16$
  - $V_2 = 4^3.3^4 = 5184 = 16 \bmod 19$
  - since $16 = 16$ signature is valid

Schnorr Digital Signatures

- Uses exponentiation in a finite (Galois)
  - Security based on discrete logarithms, as in D-H
- Minimizes message dependent computation
  - multiplying a 2*n-bit* integer with an *n-bit* integer
- Main work can be done in idle time
- Have using a prime modulus *p*
  - $p-1$ has a prime factor *q* of appropriate size
  - typically *p* 1024-bit and *q* 160-bit numbers

Schnorr Key Setup

- choose suitable primes p , q
- choose a such that $a^q = 1 \bmod p$
- (a,p,q) are global parameters for all
- each user (eg. A) generates a key
  - chooses a secret key (number): $0 < s_A < q$
  - compute their **public key**: $v_A = a^{-s_A} \bmod q$

Schnorr Key Setup

- choose suitable primes p , q
- choose a such that $a^q = 1 \bmod p$
- (a,p,q) are global parameters for all
- each user (eg. A) generates a key
  - chooses a secret key (number): $0 < s < q$
  - compute their **public key**: $v = a^{-s} \bmod p$

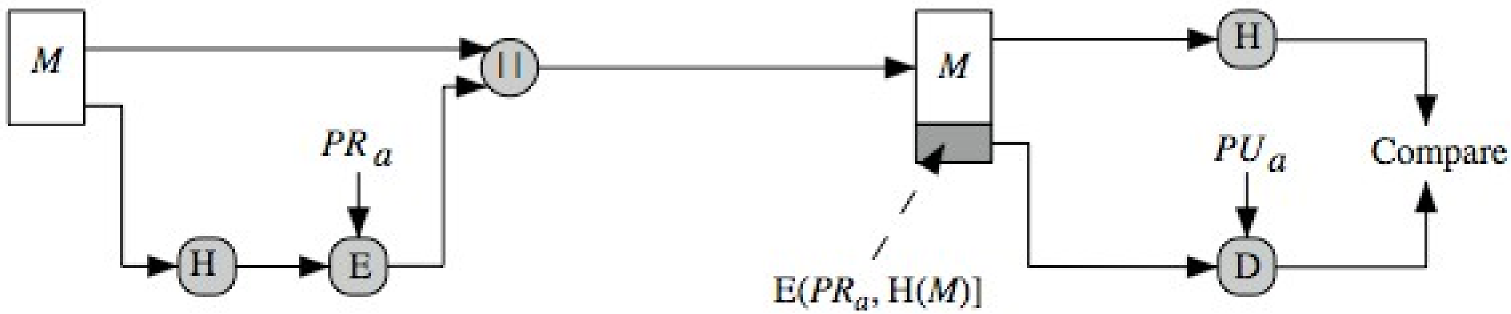
Schnorr Signature

- user signs message by
  - choosing random r with $0<r<q$ and computing $x = a^r \bmod p$
  - concatenate message with x and hash result to computing: $e = H(M \| x)$
  - computing: $y = (r + se) \bmod q$
  - signature is pair (e, y)
- any other user can verify the signature as follows:
  - computing: $x' = a^y v^e \bmod p$
  - verifying that: $e = H(M \| x')$

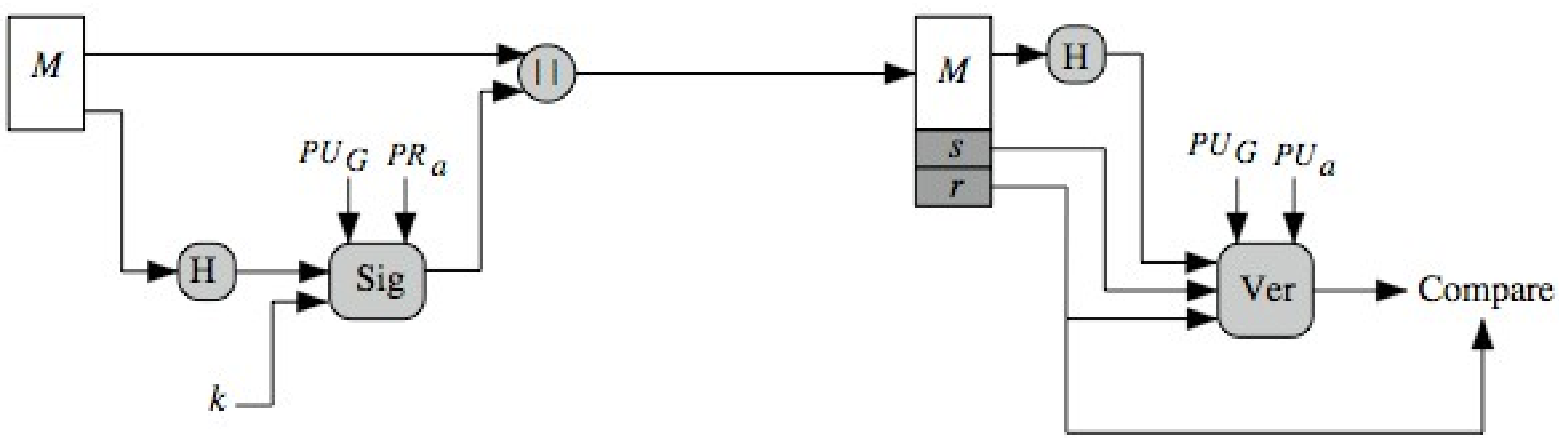
Digital Signature Standard (DSS)

- US Govt approved signature scheme
- Designed by NIST & NSA in early 90's
- Published as FIPS-186 in 1991
- Revised in 1993, 1996 & then 2000
- Uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm
- FIPS 186-2 (2000) includes alternative RSA & elliptic curve signature variants
- DSA is digital signature only unlike RSA
- is a public-key technique

DSS vs RSA Signatures



(a) RSA Approach



(b) DSS Approach

Digital Signature Algorithm (DSA)

- Creates a 320 bit signature
- with 512-1024 bit security
- Smaller and faster than RSA
- A digital signature scheme only security depends on difficulty of computing discrete logarithms
- It is a variant of ElGamal & Schnorr schemes

Main Idea

- Works in subgroup of a larger finite field.
- Works over a large finite field Z_p. p: 1000 bits long.
- Maximum size of the cyclic group = p-1.
- We will ensure that p-1 has a large prime factor q (160 bit long). Hence q divides (p-1).
- We will choose a generator of the subgroup (g).
- Then $g^{(q)} = 1 \bmod p$.
- Now we can redefine ElGamal idea over the subgroup:
  - Signing equations involve modulo q
  - Verifications are over mod p;
- DSA follows a similar strategy with some modifications.

DSA Key Generation

- have shared global public key values (p,q,g):
  - choose 160-bit prime number q
  - choose a large prime p with $2^{L-1} < p < 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
    - such that q is a 160 bit prime divisor of $(p-1)$
  - choose $g = h^{(p-1)/q}$
    - where $1<h<p-1$ and $h^{(p-1)/q} \bmod p > 1$
- users choose private & compute public key:
  - choose random private key: $x<q$
  - compute public key: $y = g^x \bmod p$

DSA Signature Creation

- to **sign** a message `M` the sender:
  - generates a random signature key `k`, `k<q`
  - nb. k must be random, be destroyed after use, and never be reused
- to **sign** then computes signature pair:

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

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

sends signature `(r,s)` with message `M`

DSA Signature Verification

- having received M & signature `(r,s)`
- to **verify** a signature, recipient computes:

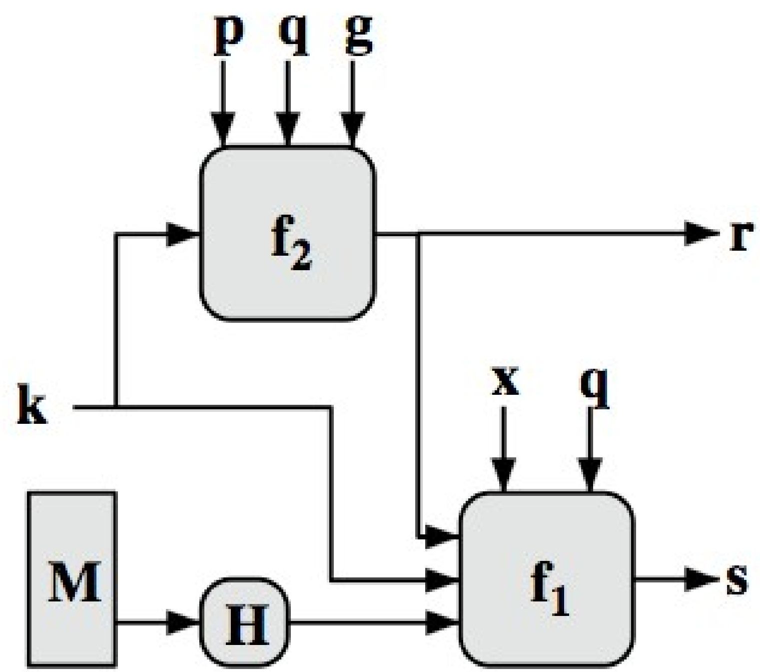
  $w = s^{-1} \bmod q$

  $u1 = [H(M)||w] \bmod q$

  $u2 = (rw) \bmod q$

  $v = [(g^{u1} y^{u2}) \bmod p] \bmod q$
- if v=r then signature is verified
- Appendix A of Chapter 13 for details of proof why

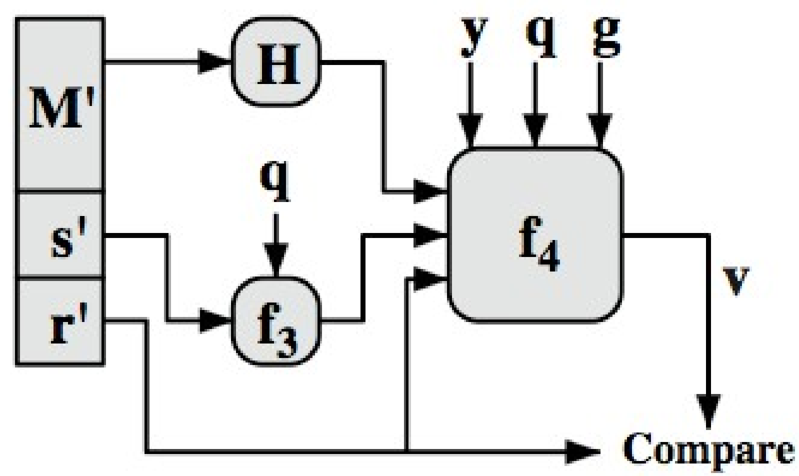
DSS Overview



$$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$$

$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$

(a) Signing



$$w = f_3(s', q) = (s')^{-1} \bmod q$$

$$v = f_4(y, q, g, H(M'), w, r')$$

$$= ((g^{(H(M')w) \bmod q}\, y^{r'w \bmod q}) \bmod p) \bmod q$$

(b) Verifying