6.857 Computer and Network Security **Fall Term, 1997**

Lecture 8 : September 30, 1997

Lecturer: Ron Rivest *Scribe: Debajit Ghosh*

1 Topics Covered

Public Key Cryptography

- Review of ElGamal
- DSS
- RSA
- Elliptic Curves

We began the course with a discussion and review of private key cryptography and then started moving into the realm of public key cryptography. During this lecture, we will discuss four public key schemes. Even though RSA preceded the other schemes presented here, we will review ElGamal and introduce DSS first (both *mod* p schemes), in hopes of providing more background and motivation for some of the concepts utilized in RSA. Finally, we will also provide a brief introduction to the theory of Elliptic Curves. Throughout this lecture, we will primarily consider public key cryptography as it relates to digital signatures.

2 ElGamal [Taher ElGamal, 1985]

This algorithm is non-deterministic, allowing for multiple valid signatures for a given message. Like other algorithms, this algorithm's security depends on the difficulty of the discrete log problem.

- First, find a large prime **p** and a generator **g** of Z_p^* (you can find one if you know the factorization of p-1).

- Pick an $x \in Z_{p-1}$ as the private key
- Compute $y = g^x (\text{mod } p)$ as the public key

To sign a message M: (for now assume $|M| < |p|$ – in the future, signing only the hash of the message will assure us of this).

- Pick random $k \in_R$ (element at random) Z^*_{p-1} (so, gcd(k,p-1) = 1) - this makes ElGamal non-deterministic
- Generate the signature σ:
  - $a = g^k (\text{mod } p)$
  - $b = \frac{(M - ax)}{k} (\text{mod } p - 1)$ (we can do this because gcd(k,p-1)=1)
  - $\sigma(M) = (a, b)$

To verify a signature $\sigma = (a, b)$:

- Check that $y^a a^b = g^M (\text{mod } p)$

3 Digital Signature Standard (DSS) [proposed by NIST, 1991]

This is a fairly interesting, non-deterministic cryptography scheme which was proposed by the government. Accordingly, there has also been some criticism and skepticism surrounding this scheme, because part of the proposal had the government "supply" the large prime p. Some believed that this could lead to a weakness in the scheme because a p could be chosen such that one could take discrete logs for that p. Or, other weaknesses in a specific p could be exploited as a "back-door" mechanism, should the p be a "weaker" prime. This is not very likely, however. Another interesting fact about DSS is that it is designed specifically for signatures, making using DSS for any kind of encryption more difficult (but modification of DSS into an encryption scheme is possible).

DSS starts off similarly to ElGamal.

- Start off by finding a large (public) prime p. This time, however, do *not* find a generator for p. Instead,
- Find another prime q | p-1, where $|q| = 160$ bits. (Often, you find q first and then guess at various p's until you find one that works.)
- Find a (public) generator g of order q modulo p ($g^q = 1 \bmod p$). This need only generate a subgroup, however. Specifically, we want $g^i \neq 1$ if $0 \leq i < q$.
- Now, pick x s.t. $0 \leq x \leq q - 1$ as the public key.
- Compute $y = g^x (\bmod\ p)$ as the private key.

To sign a message M:

- Pick $k \in_R Z_q^*$
- $a = g^k (\bmod\ p)(\bmod\ q)$. This has the advantage of reducing the message size even further. If $|p| = 1000$, but $|q| = 160$, we now have a shorter signature.
- $b = \frac{m+ax}{k} (\bmod\ q)$ (NOTE: here we use a +, not a -, as in ElGamal).
- $\sigma(M) = (a, b)$

To verify a signature:

- Check that $y^{\frac{a}{b}} g^{\frac{M}{b}} (\bmod\ p)(\bmod\ q) = a$

  $g^{\frac{ax}{b}} g^{\frac{M}{b}} (\bmod\ p)(\bmod\ q)$
  $= g^{\frac{M+ax}{b}} (\bmod\ p)(\bmod\ q)$
  $= g^k (\bmod\ p)(\bmod\ q)$
  $= a\surd$

How can we find a generator?

- Know p
- Find h s.t. $h^{p-1} \equiv 1 (\bmod\ p)$.
- So, $g = h^{\frac{p-1}{q}}$ – of order q

Again, in this scheme, the security rests on the difficulty in finding and computing discrete logarithms. Also, as before, releasing k compromises the security of the signature. Finally, DSS appears to have the advantages of computing shorter signatures, not being suitable for encryption, ease in finding generators, and having generators whose order is prime. Throughout all of this, we assume we use a short M. However, we can also use H(M), the hash of M (using MD5, etc.), and just sign the shorter hash of the message.

4 RSA [Rivest, Shamir, and Adleman, 1977]

Unlike the above schemes, RSA is based on a composite modulus. Let us try using a prime modulus p, as before, and try to develop some motivation for why we might want a composite modulus.

Consider:

$$\begin{aligned} C &= M^e (\text{mod } p) \\ C^d &= M^{ed} (\text{mod } p) \\ &= M^{k(p-1)+1} (\text{mod } p) \\ &= M^{(p-1)k} \cdot M (\text{mod } p) \\ &= 1 \cdot M (\text{mod } p) \\ &= M (\text{mod } p) \end{aligned}$$

(Let $gcd(e, p-1) = 1$, so $e^{-1} (\text{mod } p-1)$ exists.)
(Let $d = e^{-1} (\text{mod } p-1)$)

This works well privately, but it simply does not work well as a public key cryptography mechanism. If you know p and e, you can get p-1 easily and hence can compute e^{-1} to decrypt the message. So, how can we make this secure? One possibility is to make p secret. Instead of using *mod p*, how about *mod n*?

$C = M^e (\text{mod } n)$ (Let n=pq, gcd(e,p-1) = 1, gcd(e,q-1) = 1.)

When thinking about doing computations mod n, it is helpful to think about coordinate representations and residues.

$Z_n \cong Z_p \times Z_q$

e.x.: p = 5, q = 7

We can fill in a residue table as follows: ...

	0	1	2	3	4	5	6
0	0					5	
1		1					6
2	7		2				
3		8		3			
4			9		4		

... until we get the complete table:

	0	1	2	3	4	5	6
0	0	15	30	10	25	5	20
1	21	1	16	31	11	26	6
2	7	22	2	17	32	12	27
3	28	8	23	3	18	33	13
4	14	29	9	24	4	19	34

In this table, we can easily find the value of a number x mod n. First, we compute what x is mod p, as well as mod q, and then looking up the number in the above table according to the "coordinates" from the mod p and mod q calculations. For example:

x mod n		(x mod p, x mod q)
23	$\longleftrightarrow$	(3,2)
2	$\longleftrightarrow$	(2,2)
46	$\longleftrightarrow$	(6,4)=(1,4)

$M^e \longleftrightarrow (M^e(\text{mod } p), M^e \text{ mod } q)$

This of course, relates to the Chinese Remainder Theorem (covered in the readings) and how one can convert a number back and forth from the two representations (mod n and (mod p, mod q)):

$r(\text{mod } n) \longleftrightarrow (s(\text{mod } p), t(\text{mod } q))$
where s = r mod p, t = r mod q.

and

$r = (q(q^{-1}((\text{mod } p))) \cdot s + (p \cdot (p^{-1}(\text{mod } q)) \cdot t))$

So, it is relatively straightforward to convert back and forth between these representations.

With this background, we can now proceed to the RSA algorithm.

- Pick large (private) primes p,q.
- Pick small integer e s.t. gcd(e,(p-1)×(q-1)) = 1. (Use something small, such as $2^{16} + 1$, so encryption is fast.)
- Pick $d = e^{-1}(\text{mod } p-1) \times (q-1)$. Also, we know that $d = e^{-1}(\text{mod } p-1)$ and $d = e^{-1}(\text{mod } q-1)$.
- So,

$$\begin{aligned} C &= M^e(\text{mod } n) \\ C^d &= M^{ed}(\text{mod } n) \\ &= M^{k(p-1)(q-1)+1}(\text{mod } n) \\ &= M^{(p-1)(q-1)} \cdot M(\text{mod } n) \\ &= 1 \cdot M(\text{mod } n) \\ &= M(\text{mod } n) \end{aligned}$$

- Thus, we can leave p, q, d private but make n=pq, e public. We use (n,e) as our public key and d as our private key (which depends on the privacy of p,q in order to keep an adversary from computing d from e, which is possible with knowledge of p-1, q-1).
- $C = M^e(\text{mod } n)$, $M = C^d(\text{mod } n)$
- To find d, we need to know p-1, q-1. This is tough - probably as tough as factoring large numbers. What if you get unlucky, and p and q are not primes? You might get p and q as Carmichael Numbers, in which case there will be more factors of n, making it somewhat easier to factor. If you do not get Carmichael Numbers, most likely you will get garbage out of trying to "decrypt"

an encrypted message, which should notify you that something is wrong with p and q.

To sign a message M:

- $\sigma = M^d (\text{mod } n) \rightarrow (M, \sigma)$

To verify:

- Check that $\sigma^e = M$

Finally, there is a potential problem with RSA. Namely, RSA adheres to a multiplicative property. Suppose you have simple messages M_1=2, M_2=3. Then,

$$\begin{array}{rr} & E(M_1) = 2^d (\text{mod } n) \\ \times & E(M_2) = 3^d (\text{mod } n) \\ \hline & (2 \cdot 3)^d (\text{mod } n) \end{array}$$

Hence, you now know 6^d = E(6)! In practice, however, you typically sign the *hash* of the message, which scrambles up the bits and keeps this from being a real problem.

Because RSA depends on the difficulty of factoring, one must consider exactly how difficult factoring is. To date, most of the best algorithms are on the order of $\sim e^{c\ (ln\ n)^{\frac{1}{3}} (ln\ ln\ n)^{\frac{2}{3}}}$. One can actually do slightly better; the best known algorithm is the *Number Field Sieve* (which sacrifices some from the $\frac{1}{3}rd$ term in order to lower and improve the $\frac{2}{3}rds$ term). Also, while it is known that if you can factor a large number, you can break RSA, it is *not* known (or proven) that decrypting a message means that you can factor a large number.

5 Elliptic Curves [Victor Miller, Neil Koblitz]

Elliptic Curves provide another foundation upon which cryptography algorithms can be built. Now, instead of working strictly with residues mod p, we deal with geometric relations. A primary advantage of elliptic curves is that through the complex geometric calculations, a more difficult (analog of the) discrete log problem exists (or,

at least, one that is not as well understood), which allows for shorter primes p. And, of course, with shorter primes p, cryptography algorithms will be somewhat faster.

Here, the prime p is public and is utilized in equations of the following format:

$y^2 \equiv x^3 + ax + b (\text{mod } p)$ (denoted Weierstrauss normal form)

In addition, the above adheres to the following constraint:

$$4a^3 + 27b^2 \neq 0 (\text{mod } p)$$

Elliptic Curves thus consist of the points:
(x,y) : $y^2 \equiv x^3 + ax + b (\text{mod } p)$, which is symmetric with respect to the x-axis.

In addition, we add an extra point, I, which represents the point at *infinity*.

Thus, we have a **finite** set of solutions to this equation, with $p + t + 1$ pairs of points, where $|t| \leq 2\sqrt{p}$. In other words, points on an elliptic curve form a finite abelian set and can thus be used in a similar manner to fields and the such. We will now define the basic operations on this set.

Addition is defined as follows: P + Q, where P and Q are points anywhere on the curve, is defined to be the *reflection* of the third point on the curve touched by the line connecting P and Q (essentially, utilizing tangents and chords; see Figure 1). If P and Q are the same point (say, P=Q=R), then the addition is defined to be [2]R (or, more simply, 2R), where the line "connecting" R is simply the tangent to the curve at R. Since I is at infinity, the line connecting P and I is vertical and simply crosses the elliptic curve at P's reflection (-P). Thus, the reflection of this point is merely P; effectively, I is the identity function for this finite set. Equivalently, if the line connecting points P and Q is vertical (Q=-P), their sum is defined to be I, as this line "touches" the elliptic curve again at the point at infinity.

More formally, we define addition using geometry, calculating the equation for the line between P and Q as $y = \lambda x + \beta$, find the third point of intersection between the elliptic curve and this line, and subsequently take the reflection of that point as the sum. The following algorithm outlines this procedure:

- Given P = (x_1,y_1), Q = (x_2,y_2),
  if $(x_1 = x_2) \wedge (y_1 = -y_2)$ then P+I := I (Q=-P).
  if $(x_1 = x_2)$ then $\lambda = \frac{3x_1^2 + a}{2y_1} (\text{mod } p)$, (where λ is the slope. i.e., this formula is obtained by taking the derivative of the curve at the point (x_1,y_1)).

else $\lambda = \frac{y_2 - y_1}{x_2 - x_1} (\text{mod } p)$

- Define $\beta = y_1 - \lambda x_1$.
- Then, $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = -(\lambda x_3 + \beta)$
- P+Q := (x_3,y_3) (mod p).

This addition operation is commutative as well as *associative* (somewhat surprising; the proof is beyond the scope of this class). Thus, we now have a system where we can define addition analogously to multiplication mod p.

We utilize the following notation with this set of numbers.

- $E_n(a, b)$ = set of points on the elliptic curve; i.e. satisfies:
  - $y^2 = x^3 + ax + b$
  - gcd($4a^3 + 27b^2$,n)=1

  plus contains the special point I.
- E_p, for prime p, represents an elliptic group
- $N_p(a, b)$ = order of $E_p(a, b)$.
- $O_{E_p}(M)$ = order of an element M in E_p.

So, now, instead of multiplication, we use the addition operation defined above. Subsequently, instead of exponentiation $g^k (\text{mod } p)$, we "multiply" P (g's equivalent in this realm) by k (= [k]P), defined as: P + P + P + P ... + P (k times). This makes the "discrete log problem" equivalent to discovering how many times P must be added to itself to equal another point Q. In summary:

Elliptic Curves		ElGamal
P+Q	$\approx$	$x \cdot y (\text{mod } p)$
$[k]P$ = P+P+P...+P (k times)	$\approx$	$g^k (\text{mod } p)$

We can make the points P and Q=[k]P public as long as we keep k secret; this is completely analogous to working mod p. However, now some discrete log "tricks" no longer work, which effectively allows us to use shorter keys, making for overall faster cryptography algorithms and implementations.

6 Handouts and References

- Handout 8: Readings for Lecture 8 (Description of MD5 from *Applied Cryptography*).

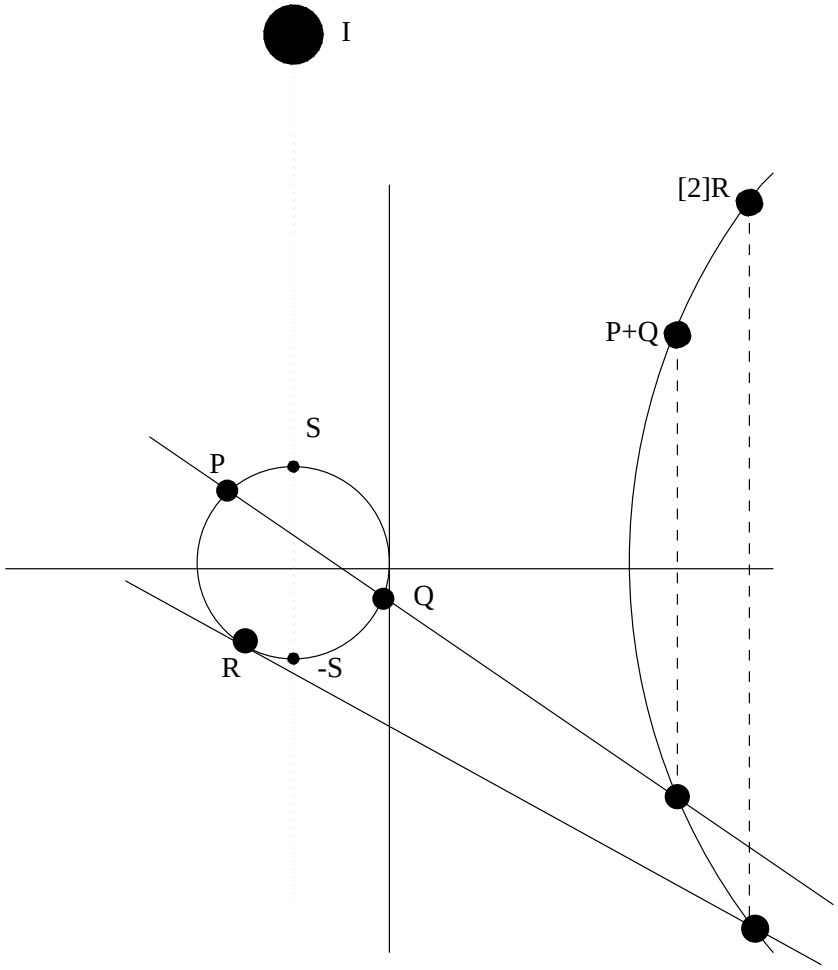

Figure 1: An elliptic curve consisting of more than one "piece." One can see the process for addition of points P and Q (as well as R + R) illustrated above. Also note special point I and the vertical line connecting it and any points S and -S.