Chosen plaintext attack on NDS January 15

Goal: find S_k.

Exhaustive keysearch is infeasible. 2048 bits $\rightarrow 2^{2048}$ possible keys.

Idea: Let T denote one round of encryption, i.e. takes a message in two halves

$$T(m_{i-1}, m_i) - (m_i, m_{i-1} \oplus f(m_i). \tag{1}$$

Let F denote the encryption function. (All 16 rounds.)

Main observation: $F = T^{16}$.

So, $T(F(m)) = T(T^{16}(m)) = T^{17}(m) = F(T(M))$. i.e. F and T commute.

Attack: (with justification)

- Fix $r \in \{0,1\}^8$. We will determine $S_k(r)$.
- Select $u = (m_0, m_1)$ such that
$$m_1^* = r \tag{2}$$
and so
$$S_0(n_i^j) \neq S_1(n_2^j), \quad \forall\ 1 \leq j \leq 8 \tag{3}$$
- Obtain encryption of $u : F(u) = (a, b)$. Then
$$\begin{aligned} T(F(u)) = T(a,b) &= (b, ?) \qquad \text{by (1)} \\ &= F(T(u)) \end{aligned}$$
- Select a byte $t \in \{0,1\}^8$ and guess that $S_k(r) = t$.
- Apply one round of encryption to u, assuming that $S_k(r) = t$. Call the result $T_t(u)$.
- Obtain encryption $F(T_t(u)) = (c, d)$.

  Note:

  - if $S_k(r) = t$, then $T_t(u)$, so $F(T_t(u)) = (b, ?)$, so $b = c$.
  - if $S_k(r) \neq t$, then $T_t(u) \neq T(u)$ by (3).

Now, if we make the heuristic assumption that F behaves like a random permutation, then the probability that one would encrypt $F(T_t(u)) = F(T(u)) = (b, ?)$ is 2^{-64} which is negligibly small. So if $b \neq c$ change t and repeat. Otherwise $S_k(r) = t$; correct with high probability.

Chosen plaintext attack on NDS (Continued) January 15

Attack:

procedure CHOSEN PLAINTEXT ATTACK ON NDS
  for $r \in \{0,1\}^8$ **do**
    Select $u = (m_0, m_1)$ such that $m_1^* = r$ and $S_0(n_1^j) \neq S_1(n_2^j)$ for all $j, 1 \leq j \leq 8$.
    Obtain $F(u) = (a, b)$.
    for $t \in \{0,1\}^8$ **do**
      Compute $T_t(u)$
      Obtain $F(T_t(u)) = (c, d)$
      if $b = c$ **then**
        Conclude $S_k(r) = t$
        goto next r
      end if
    end for
  end for
end procedure

Analysis: Expected number of chosen plaintexts is:

$$256(1 + 128) \approx 2^{15} \approx 320,000 \ll 2^{2048}.$$

Therefore NDS is *totally* insecure.

Finding collisions. January 24

Let $H : \{0,1\} \to \{0,1\}^n$ be a hash function.

PROBLEM: Find $x_1, x_2 \in \{0,1\}^*, x_1 \neq x_2$ with $H(x_1) = H(x_2)$.

Recall the naïve method has run time

$$\sqrt{\frac{\pi N}{2}}$$

where $N = 2^n$, and storage requirements

$$\sqrt{\frac{\pi N}{2}} * 2n \text{ bits.}$$

E.g. if $n = 128$, then time is $\approx 2^{64}$ steps, but the storage is $7 * 10^8$ terabytes. (About 700 billion dollars.)

Von Oorschol - Wiener (VW) collision search (1993)

Define a sequence $\{x_i\}$ by $x_0 \in_R \{0,1\}^n$ and $x_i = H(x_{i-1}), \forall i > 1$. Let y be the smallest index for which $x_i = x_j$ for some $i < j$. (Such a j must exist.)

Then $x_{i+\ell} = x_{j-1}$) is a collision for H.

By the birthday paradox,

$$E[j] = \sqrt{\frac{\pi N}{2}},$$

$$\textbf{FACT}: \quad E[i] = \frac{1}{2}\sqrt{\frac{\pi N}{2}}, \quad \text{and}$$

$$E[j-i] = \sqrt{\frac{\pi N}{2}}.$$

MAIN IDEA: Store *some* of the x_i's.

Define a *distinguishing property* of elements of $\{0,1\}^n$ (e.g. first 10 bits are all 0). Let θ be the proportion of elements in $\{0,1\}^n$ that are distinguished.

procedure STAGE 1 ▷ Detecting a collision.
  Select distinguished point $x_0 \in \{0,1\}^n$.
  660 Store $(x_0, 0, -)$ in a table sorted by first entry.
  $LP \leftarrow x_0$. ▷ LP is last point stored.
  for $j = 1, 2, \ldots$ **do**
    compute $x_j = H(x_{j-1})$.
    if x_j is distinguished **then**
      store (x_j, j, LP).
      update $LP \leftarrow x_j$.
      if $x_i = x_j, i < j$ **then**
        goto STAGE 2.
      end if
    end if
  end for
end procedure

procedure STAGE 2 ▷ Finding a collision, see fig. 3 handout.

```
    ℓ1 ← i − a, ℓ2 ← j − b.
    ensure ℓ1 ≥ ℓ2.
    k ← ℓ1 − ℓ2.
    for m=1,2,... do
        compute (x_{a+k+m}, x_{b+m})
        if x_{a+k+m} = x_{b+m} then
            return (x_{a+k+m−1}, x_{b+m−1}).
        end if
    end for
end procedure
```

- Analysis

  Stage 1 expected time:

  $$\sqrt{\frac{\pi N}{2}} + \frac{1}{\theta}$$

  Stage 2 expected time:

  $$\frac{3}{\theta}.$$

  Expected storage:

  $$\theta + \sqrt{\frac{\pi N}{2}}(3n) \text{ bits}$$

Parallelizing the VW collision finding algorithm January 27

Given m-processors, run the algorithm independently on all processors. Keep the database of distinguished points on one global server.

Expected time:

$$\frac{1}{m}\sqrt{\frac{\pi N}{2}} + \frac{4}{\theta}$$

Space:

$$\theta + \sqrt{\frac{\pi N}{2}}(3n) \text{ bits}$$

(same).

Note: the processors do not communicate with each other. They only occasionally communicate with the central server.

Finding meaningful collisions.

Let m_1 and m_2 be arbitrary messages. We'll show how A (the VW algorithm) can be used to find modifications $\hat{m}_1$ and $\hat{m}_2$ of m_1 and m_2 so that $H(\hat{m}_1) = H(\hat{m}_2)$ and $\hat{m}_1$ and $\hat{m}_2$ have the same meaning as m_1 and m_2, respectively. The new algorithm will have the same run-time as well.

- Define $g_{m_1} : \{0,1\}^n \to \{0,1\}^*$ as follows:

  Fix n positions in m_1, say $j_1, \ldots, j_n$ (e.g. the ends of sentences). Then for $r \in \{0,1\}^n$, define $g_{m_1}(r)$ to be the message obtained from m_1 by adding for each $1 \leq i \leq n$ a space at position j_i. iff the i-th bit of r is 1. Note $g_{m_1}(r)$ has the same meaning as m_1.

- Define g_{m_2} in the same way.

- Partition $\{0,1\}^n$ into 2 sets S_1 and S_2 of the same size.

- Define $f : \{0,1\}^n \to \{0,1\}^n$ as follows:

$$f(r) = \begin{cases} H(g_{m_1}(r)) \text{ if } r \in S_1 \\ H(g_{m_2}(r)) \text{ if } r \in S_2 \end{cases}$$

  If H is viewed as a random function, then so is f.

  Attack: Run A on f.

  A will produce a collision (a, b). Suppose a and b are from different sets S_1 and S_2, i.e. $a \in S_1$, $b \in S_2$. Then

$$f(a) = H(g_{m_1}(a)) = H(g_{m_2}(b)).$$

  Letting $\hat{m}_1 = g_{m_1}(a), \hat{m}_2 = g_{m_2}(b)$, we have $H(\hat{m}_1) = H(\hat{m}_2)$

Iterated Hash Functions January 29

Ingredients:

- Compression function $f : \{0,1\}^{n+r} \to \{0,1\}^n$.
- Initialization vector $\text{IV} \in \{0,1\}^n$.

To hash $x \in \{0,1\}^*$, do:

- Break x into r-bit blocks: $\hat{x} = x_1, \ldots, x_t$, where x_t is padded with zeros if necessary.
- Let b be the bit length of x.
- Let x_{t+1} be the (right-justified) binary representation of b.
- So $H : \{0,1\}^* \to \{0,1\}^n$.

Theorem: (Merkel) If f is collision resistant, then H is collision resistant.

Proof. Suppose H is not collision resistant. Then we have an efficient algorithm which finds a collision for H. We use the algorithm to find a collision for H; (x, x'). Write:

$$\hat{x} = x_1, x_2, \ldots, x_t, b = \text{ bitlength of } x, x_{t+1}, \text{length block of } x$$
$$\hat{x}' = x'_1, x'_2, \ldots, x'_t, b' = \text{ bitlength of } x', x'_{t+1}, \text{length block of } x'$$

Now, efficiently compute

$$\begin{aligned} H_0 &= \text{IV} \\ H_1 &= f(H_0, x_1) \\ &\vdots \\ H_t &= f(H_{t-1}, x_t) \\ H_{t+1} &= f(H_t, x_{t+1}) \end{aligned} \qquad \begin{aligned} H'_0 &= \text{IV} \\ H_1 &= f(H_0, x_1) \\ &\vdots \\ H_t &= f(H_{t-1}, x_t) \\ H_{t+1} &= f(H_t, x_{t+1}) \end{aligned}$$

So $H(x) = H(x')$
If $b \neq b'$ then $x_{t+1} \neq x'_{t'+1}$. So $((H_t, x_{t+1}), (H'_{t'}, x'_{t'+1}))$ is a collision for f.
If $b = b'$ then $t = t'$. Then $H_t, H'_{t'}$ might be a collision if they are not equal. If they are equal, try $(H'_{t-1}, H'_{t'-1})$. The collision is the largest i such that $(H_i, x_{i+1}) \neq (H'_i, x'_{i+1})$ but $f(H_i, x_{i+1}) = f(H'_i, x'_{i+1})$. □

Using a MAC Scheme for Non-Repudiation February 5

A MAC Scheme and trusted third party can be used to verify that Alice signed a message:

1. Alice shares a key k_{AT} with a trusted third party (TTP).
2. Alice sends $(x, \mathrm{MAC}_{k_{AT}}(x))$ to Bob.
3. Bob asks the TTP to verify the signed message.
4. In case of a dispute, the judge asks the TTP to verify the signed message.

Basic RSA Encryption Scheme February 12
Key Generation.

1. Randomly select p, q distinct large primes of bitlength $k \geq 512$.
2. Compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$.
3. Select $e \in [1, \varphi(n)]$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d = e^{-1} \mod \varphi(n)$.
5. Public key is (n, e) and private key is d.

Encryption.

1. Obtain an authentic copy of Bob's public key, (n, e).
2. Represent the message as an integer $m \in [0, n-1]$.
3. Compute $c = m^e \pmod{n}$.
4. Send c to Bob.

Decryption.

1. Compute $m = c^d \mod n$.

Proof that RSA Decryption Works. Since $ed \equiv 1 \pmod{\varphi(n)}$, we can write $ed = 1 + k\varphi(n)$ for some $k \in \mathbb{Z}$. If $m \equiv 0 \pmod{p}$, then $m^{ed} \equiv 0 \pmod{p}$, so $m^{ed} \equiv m \pmod{p}$. If $m \not\equiv 0 \pmod{p}$, then $m^{p-1} \equiv 1 \pmod{p}$ (by Fermat). Thus $m^{k(p-1)(q-1)} \equiv 1 \pmod{p}$ and thus $m^{k(p-1)(q-1)+1} \equiv m \pmod{p}$. Thus $m^{ed} \equiv m \pmod{p}$. Similarly $m^{ed} \equiv m \pmod{q}$, so $m^{ed} \equiv m \pmod{n}$, since p and q are distinct primes.

Basic RSA Signature Scheme
Signature Generation.

1. Compute $M = H(m)$ where H is a hash function.
2. Compute $s = M^d \pmod{n}$.
3. Send message-signature pair (m, s).

Signature Verification.

1. Obtain an authentic copy of public key (n, e).
2. Compute $M' = H(m)$
3. Accept message if $s^e \equiv M' \pmod{n}$.

Security of RSA February 14

I. Key generation

II. Encryption

III. Signatures

I. Key Generation

Given a public key (n, e), can an adversary obtain the private key (n, d)?

- COMPUTE-d: given (n, e), compute d.
- FACTOR: given (n, e), compute p and q.
- COMPUTE-$\varphi(n)$: given (n, e), compute $\varphi(n)$.

Definition: Let A_1 and A_2 be two computational problems. We say that A_1 *polynomial time reduces to* A_2, written $A_1 \leq_p A_2$ if there is a polynomial time algorithm for solving A_1 which uses a polynomial time oracle for solving A_2. If $A_1 \leq_p A_2$ and $A_2 \leq_p A_1$ then we say A_1 and A_2 are *computationally equivalent* and write $A_1 \equiv_p A_2$.

E.g. COMPUTE-$\varphi(n) \leq_p$ FACTOR.

Proof. We are given an oracle for FACTOR and an instance (n, e) of COMPUTE-$\varphi(n)$. We need to compute $\varphi(n)$. We give (n, e) to FACTOR to obtain p, q. Now we can compute $\varphi(n) = (p-1)(q-1)$. All of this is done in polynomial time. □

Fact: FACTOR $\leq_p$ COMPUTE-d.

II. Encryption

- What does it mean for RSA encryption to be secure?
- RSAP: Given (n, e) and c, find $m \in [0, n-1]$ such that $c = m^e \pmod{n}$.
- Clearly RSAP $\leq_p$ FACTOR. Does FACTOR $\leq_p$ RSAP? This is a big question.
- Chosen ciphertext attack on RSA encryption:
  - E is given (n, e) and target ciphertext $c = m^e \pmod{n}$.
  - E has access to a decryption oracle, to which E can present any ciphertext except c itself.

  Attack:

  (i) E selects arbitrary $x \in [2, n-1]$ with $\gcd(x, n) = 1$.

  (ii) E computes $\hat{c} = cx^e \pmod{n}$ and presents $\hat{c}$ to the decryption oracle which returns $\hat{m} = \hat{c}^d \pmod{n}$.
  Note: $\hat{m} = \hat{c}^d = (cx^e)^d = c^d x^{ed} = mx \pmod{n}$.

  (iii) E computes $m = \hat{m}x^{-1} \pmod{n}$.
- To circumvent the attack, Alice adds some "structure" to m prior to decryption. Now, if a ciphertext is decrypted by Alice and the resulting plaintext does not have the required structure, the ciphertext is rejected.
- Forward search attack on basic RSA encryption:
  - Suppose the plaintext space is small or easily predictable.
  - E can build a dictionary of (PT, CT) pairs and then compare the transmitted ciphertext with the ones in the dictionary.
  - To circumvent the attack, the plaintext is "salted", i.e. a random string of length 128 bits (say) is appended to m prior to encryption.

Security of RSA continued February 26

- Definition: a public key encryption scheme is *secure* if it is symantically secure against chosen ciphertext attack, by computationally bounded adversary.
- Basic RSA is *insecure*.
- In practice RSA PKCS #1 V1.5 is used.
  - Encryption: $M \mapsto m \mapsto c = m^e \pmod{m}$, where m is the formatted version of M.
  - Decryption: $c \mapsto m = c^d \pmod{n} \mapsto M$, where we check and remove the formatting of m to obtain M.
- Let k be the bitlength of n (e.g. $k = 128$).
- Formatting:

00	02	$xxx \ldots xxx$	00	M

- **Problem:** RSA PKCS #1 V1.5 is *not* secure.
- Blerchenbacher (1998) "restricted" chosen-ciphertext-attack on RSA PKCS #1 V1.5.
- Suppose c is the target ciphertext. The adversary can carefully select about 500,000 variants c' of c and presents c' to the decryption. The decryptor lets the adversary know if the corresponding plaintext is formatted correctly or not.
- Also used is RSA-OAEP (PKCS#1 V2). RSA-OAEP is *secure*.

III. **RSA PKCS #1 V1.5 Signature Scheme**

Signature Generation. Alice does:

(a) Compute $h = H(m)$, where H is a hash function.

(b) Format h as follows where k is the bytelength of n

$$M = \underbrace{\texttt{0001FFFF}\cdots\texttt{FF00}\overbrace{xxxx\cdots xx}^{\text{hash name}}\texttt{h}}_{k \text{ bytes total}}$$

(c) Compute $s = M^d \mod n$.

(d) The signed message is (m, s).

Signature Verification. Bob does:

(a) Obtain an authentic copy of Alice's public key.

(b) Compute $M = s^e \mod n$.

(c) Check the formatting of M:

- Write M as a bytestring of length k.
- Check:
  - 1st byte is `00`
  - 2nd byte is `01`
  - Next bytes are `FF` bytes followed by a `00` byte.

(d) Extract hash name from next 15 bytes.

(e) Extract h from the remaining bytes (length based on hash function) and check there are no extra bytes at the end.

(f) Accept iff $h = H(m)$.

Breaking RSA PKCS #1 V1.5 Signatures by Hand.

Assumptions:

(a) $e = 3$ (this is widely used).

(b) Verifier does not check for extra "garbage" bytes at end of M. (In 2006: OpenSSL, Firefox 2, Sun's JRE, Adobe Acrobat, etc did not check.)

(c) $H =$ SHA-1 (WLOG)

(d) n is 3072 bits (WLOG)

Attack:

(a) Select any m.

(b) Compute $h = H(m)$.

(c) Let $D = \texttt{00}\underbrace{xxxx\cdots xx}_{\text{hash name}}h$

(d) Let $N = 2^{288} - D$ and assume that $3 \mid N$ (if not, modify m and repeat).

(e) Let $s = 2^{1019} - 2^{34}N/3$.

Claim: (m, s) is accepted by the verifier.

Proof:

$$\begin{aligned} s^3 &\equiv (2^{1019} - 2^{34}N/3)^3 \mod n \\ &= 2^{3057} - 2^{2072}N + \underbrace{\frac{2^{1037}N^2}{3} - \left(\frac{2^{34}N}{3}\right)^3}_{\text{garbage} < 2^{2072}} \\ &= 2^{3057} - 2^{2072}(2^{288} - D) + \text{garbage} \\ &= 2^{3057} - 2^{2360} + 2^{2072}D + \text{garbage} \\ &= 2^{2360}(2^{697} - 1) + 2^{2072}D + \text{garbage} \end{aligned}$$

$$M_{\text{hex}} = \texttt{0001}\underbrace{\texttt{FFFFFF}\cdots\texttt{FF}}_{\text{696 bits}}\underbrace{\texttt{00}xxxx\cdots xx}_{\text{288 bit } D}\underbrace{yyyyyy\cdots yy}_{\text{2071 bit garbage}}$$

Math background for discrete logs March 7, 2014

Background in number theory:

- Let p be a prime.
- $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ with addition done modulo p.
- $\mathbb{Z}_p^* = \{0, 1, \ldots, p-1\}$. (All integers relatively prime to p i.e. all of them)
- Fermat's Little Theorem:

  If p is prime and $a \in \mathbb{Z}_p^*$ then $a^{p-1} = 1 \pmod{p}$.
- Definition: Let P be prime, and $a \in \mathbb{Z}_p^*$ The *order* of a, denoted $\text{ord}(a)$ is the smallest positive integer t such that $a^t \equiv 1 \pmod{p}$.

  e.g. Let $p = 13$. In $\mathbb{Z}_{13}$:

$$\begin{aligned} \text{ord}(1) &= 1 \\ \text{ord}(3) &= 3 \\ \text{ord}(12) &= 2 \\ \text{ord}(2) &= 12 \end{aligned}$$

- **Facts.** Let p be prime, $a \in \mathbb{Z}_p^*$, and suppose $ord(a) = t$.
  1. Let $r \in \mathbb{Z}$. Then $a^r \equiv 1 \pmod{p}$ iff $t|r$.
  2. $t|p-1$
  3. Let $x, y \in \mathbb{Z}$. Then $a^x \equiv a^y$ iff $x \equiv y \pmod{t}$.
  4. If $r \in \mathbb{Z}$ then $a^r \equiv a^{r \pmod{t}} \pmod{p}$.
- Definition. Let p be a prime. Then $a \in \mathbb{Z}_p^*$ is a *generator* of $\mathbb{Z}_p^*$ if $\text{ord}(a) = p-1$. Then $\{a^i \pmod{p} : 0 \leq i \leq p-2\} = \mathbb{Z}_p^*$.

Discrete Logarithm (DL) public-key systems March 10

- Security is based on hardness of discrete log problem.
- Let p, q be primes with $q|(p-1)$. Let g be an element of order q in $\mathbb{Z}_p^*$.
- Such an element g exists:

  Let α be a generator of $\mathbb{Z}_p^*$. Then let $g = \alpha^{(p-1)/q} \pmod p$. Then $g^q = \alpha^{p-1} \equiv 1 \pmod p$, so $\text{ord}(g)|q$. Since q is prime, and $\text{ord}(g) \neq 1$, we have $\text{ord}(g) = q$.
- Definition: $\langle g \rangle = \{q^i \pmod p : a < i \leq q-1\}$, or the *subgroup of $\mathbb{Z}_p^*$generated by g.*
- E.g. Take $p = 67$, $q = 11$, $g = 9$.

$9^0 \equiv 1$	$9^3 \equiv 59$	$9^6 \equiv 64$	$9^9 \equiv 24$
$9^1 \equiv 9$	$9^4 \equiv 62$	$9^7 \equiv 40$	$9^{10} \equiv 15$
$9^2 \equiv 14$	$9^5 \equiv 22$	$9^8 \equiv 25$	$9^{11} \equiv 1$

  So $\text{ord}(9) = 11$.

  What power of 9 gives 25 modulo 67? Answer: 8, i.e. $8 = \log_9 25$.

Discrete Log Problem (DLP)
Given p, q, g and $h \in \langle g \rangle$, find the integer $a \in [0, q-1]$ such that $h \equiv g^a \pmod p$. We write $a = \log_g h$ (in $\mathbb{Z}_p^*$).
Note:

- $\log_g(h_1 h_2) \equiv \log_g(h_1) + \log_g(h_2) \pmod q$.
- $\log_g(h_1^c) = c \log_g(h_1) \pmod q$.

Fastest known algorithms for DLP

1. Exhaustive search

   Compute $g^0, g^1, \ldots \pmod p$ until you encounter h.

   Runtime: $\mathcal{O}(q)$ modulo multiplications.

2. Shanks' algorithm

   Idea: Let $m = \lceil \sqrt{q} \rceil$. Write $a = im + j$ where $0 \leq j \leq m-1$ and $0 \leq i \leq m-1$. Notice that i and j are unique.

   Then $h \equiv g^a \equiv g^{im+j} \pmod p$, so

   $$g \cdot (g^{-m})^i \equiv g^j \pmod p$$

   Same idea as meet in the middle attack on double DES ($c = DES_{k_2}(DES_{k_1}(m))$)

   Algorithm:

   1. Compute $g^j \pmod p$ for $j = 0, .., m-1$.
   Store $(g^j \pmod p, j)$ in the table, sorted by first entry

2. Compute $g^m \pmod p$ and $t = g^{-m} \pmod p$
3. For $i = 0, 1, \ldots$ compute $h \cdot t^q \pmod p$. If this is in the table, say $ht^i = g^j \pmod p$ then $\log_g h = im + j$. STOP.

Runtime: $\mathcal{O}(\sqrt{q})$ modular multiplications.

Storage: $\mathcal{O}(\sqrt{q})$.

3. Pollard's Algorithm:

   Runtime: $\mathcal{O}(\sqrt{q})$ modular multiplications.

   Storage: negligible.

4. NFS.

   Runtime $L_p[\frac{1}{3}, 1.923]$.

   Recall that runtime of NFS for factoring n is $L_p[\frac{1}{3}, 1.923]$.

E.g. For an 80-bit security leevel, we need $p \approx 2^{1024}$ and $q \approx 2^{160}$. (see runtim for NFS factoring.)

E.g. For a 128-bit security level, we need $p \approx 2^{3072}$ and $q \approx 2^{256}$.

Next time:

1. Diffie-Hellman key agreement.
2. Digital Signature Algorithm (DSA).

Diffe-Hellman (DH) Key Agreement Scheme (1975) March 14
Purpose: Two parties to agree upon a shared secret key.
Domain parameters (public): p, q, g such that p and q are primes, $q \mid p-1$, $q \in \mathbb{Z}_p^*$ with $\text{ord}(g) = q$. (For example, for 80-bit security, one might use a 160-bit q and 1024-bit p.
Unauthenticated DH:

1. Alice selects $x \in_R [1, q-1]$ and computes $X = g^x \mod p$ and sends X to Bob.
2. Bob selects $y \in_R [1, q-1]$ and computes $Y = g^y \mod p$ and sends Y to Alice.
3. Alice computes $K = Y^x \mod p = g^{yx} \mod p$ and Bob computes $K = X^y \mod p = g^{xy} \mod p$. Both Alice and Bob have the same K.
4. The shared secret key is $k = H(K)$.

An eavesdropper is faced with the following problem:

> *Diffe-Hellman Problem (DHP)*: Given $p, q, g, g^x \mod p, g^y \mod p$, compute $g^{xy} \mod p$.

Clearly DHP $\leq_P$ DLP. It is conjectured that DLP $\leq_P$ DHP.
Person-in-the-middle attack: The attacker:

- Chooses x', y'.
- Intercepts Alice's X and sends her $Y' = g^{y'} \mod p$. Alice then computes $K_1 = g^{xy'} \mod p$. The attacker can also compute K_1.
- Intercepts Bob's Y and sends him $X' = g^{x'} \mod p$. Bob then computes $K_2 = g^{yx'} \mod p$. The attacker can also compute K_2.
- Intercepts a message from Alice, decrypts it using K_1, reads it, encrypts it using K_2, and sends it to Bob. The attacker can read the messages sent from Alice to Bob without them knowing. (A similar process works for messages sent from Bob to Alice.)

Authenticated DH:

- Alice sends Bob (X, "Alice").
- Bob then sends Alice ($Y, \text{Sign}_{\text{Bob}}(Y, X, \text{"Alice"})$). ($\text{Sign}_{\text{Bob}}$ denotes Bob's RSA signature.)
- Alice then verifies Bob's signature on (Y, X, "Alice")) and sends to Bob $\text{Sign}_{\text{Alice}}(X, Y, \text{"Bob"})$. ($\text{Sign}_{\text{Alice}}$ denotes Alice's RSA signature.)

Digital Signature Algorithm (DSA) (1991) March 14
(Standardized by US government in 1993.)
Domain parameters (public): p, q, g such that p and q are primes, $q \mid p-1$, $q \in \mathbb{Z}_p^*$ with $\text{ord}(g) = q$.
Key Generation: Alice does

1. Select $a \in_R [1, q-1]$.
2. Compute $h = g^a \mod p$.
3. Alice's private key is a and her public key is h.

Signature Generation: To sign $M \in \{0,1\}^*$, Alice does:

1. Compute $m = H(M)$.
2. Select a *per-message secret* $k \in_R [1, q-1]$.
3. Compute $r = (g^k \mod p) \mod q$ and check that $r \neq 0$.
4. Compute $s = k^{-1}(m + ar) \mod q$ and check that $s \neq 0$.
5. Alice's signature on M is (r, s).

Digital Signature Algorithm (DSA) Continued March 17

Signature Verification: To verify $M, (r, s)$, Bob does:

1. Obtain an authentic copy of Alice's public key h.
2. Verify that $1 \leq r, s \leq q - 1$.
3. Compute $m = H(M)$.
4. Compute $u_1 = s^{-1}m \mod q$ and $u_2 = s^{-1}r \mod q$.
5. Compute $v = (g^{u_1}h^{u_2} \mod p) \mod q$.
6. Accept iff $v = r$

Idea: $s \equiv k^{-1}(m+ar) \pmod q \iff k \equiv s^{-1}(m+ar) \pmod q \iff g^k \equiv g^{s^{-1}(m+ar)} \pmod p$

$\iff g^k \equiv g^{s^{-1}m}h^{s^{-1}r} \pmod p \Rightarrow \underbrace{(g^k \mod p) \mod q}_{r} = \underbrace{(g^{s^{-1}m}h^{s^{-1}r} \mod p) \mod q}_{v}$

Security Notes:

1. DSA is *believed* to be *secure*, assuming DLP is hard and H is "secure".
2. k should be securely destroyed after it is used.
3. k should never be reused.
4. Finding M such that $H(M) = 0$ should be infeasible. [See assignment 4.]

DSA vs RSA Signature Schemes at 128-bit security level:

- For RSA with 128-bit security, we need n to be 3072 bits.
  For DSA with 128-bit security, we need p to be 3072 bits and q to be 256 bits.
- *Signature Size*: RSA signatures are 3072 bits, whereas DSA signatures are 512 bits. Thus, DSA signatures are better in applications where small signatures are required.
- *Speed*:
  RSA: In practice, $e = 3$ or $e = 2^{16} + 1$ so signature verification is fast. However, signature generation is slow, but one could not speed it up by choosing a small d, because this would make the scheme insecure.
  DSA: In signature generation, $k, r, k^{-1}, k^{-1}ar$ can be computed before one knows M. Signing then requires computation of $m = H(M)$ and $s = k^{-1}(m + ar)$. So signature generation is fast.

Elliptic Curve Cryptography March 17

Idea: Replace $\langle g \rangle$ with the group of points on an elliptic curve.

Definition. Let $F = \mathbb{R}$ or $\mathbb{Z}_p$ (where $p \geq 5$ is prime). An elliptic curve over F is defined by an equation $Y^2 = X^3 + aX + b$ where $a, b \in F$ and $4a^3 + 27b^2 \neq 0$. The points on E is $E(F) = \{(x, y) : y^2 = x^3 + ax + b, \ \ x, y \in F\} \cup \{\infty\}$ where ∞ is the point at infinity.

Examples.

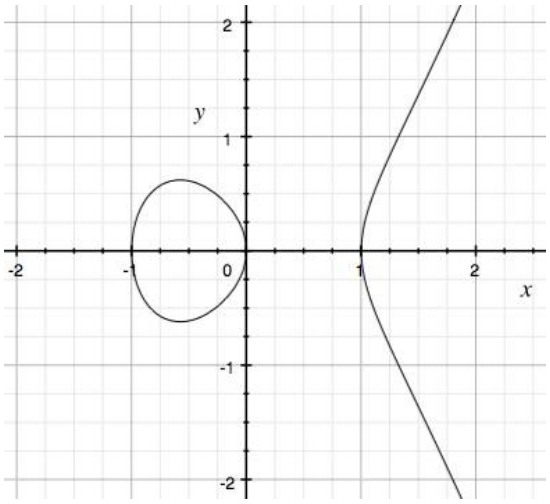

Figure 1: $E : Y^2 = X^3 - X$ over $\mathbb{R}$

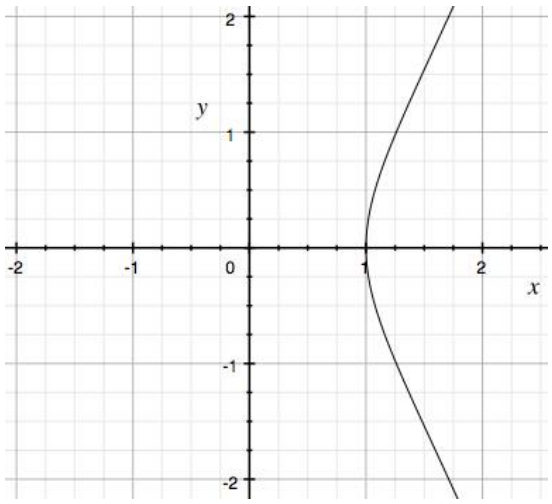

Figure 2: $E : Y^2 = X^3 - 1$ over $\mathbb{R}$

Example. $E : Y^2 = X^3 + 2X + 9$ over $\mathbb{Z}_{13}$. The points on E are:

$$E(\mathbb{Z}_{13}) = \{(0,3), (0,10), (1,5), (1,8), (3,4), (3,9), (4,4), (4,9), (5,1), (5,12), (6,4), (6,9), (8,2),$$
$$(8,11), (11,6), (11,7), \infty\}$$

Number of points in $E(\mathbb{Z}_p)$

Clearly, $1 \leq \#E(\mathbb{Z}_p) \leq 2p+1$. In fact, $(\sqrt{p}-1)^2 \leq E(\mathbb{Z}_p) \leq (\sqrt{p}+1)^2$. (Hasse's Theorem). Hence $\#E(\mathbb{Z}_p) \approx p$.

There is a natural way to add two points in $E(F)$ to get a third point in $E(F)$.

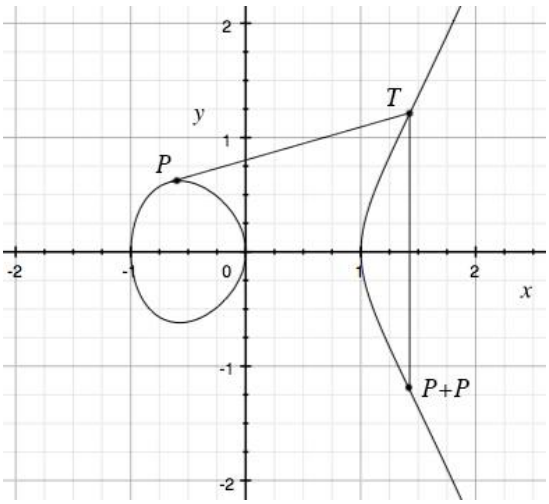

Figure 3: $P+Q$

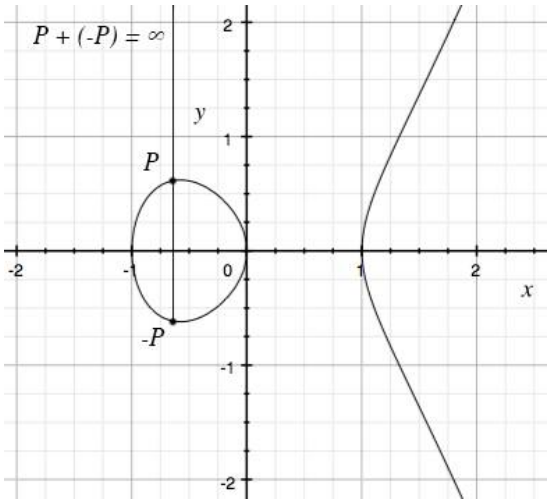

Figure 4: $P+(-P)$

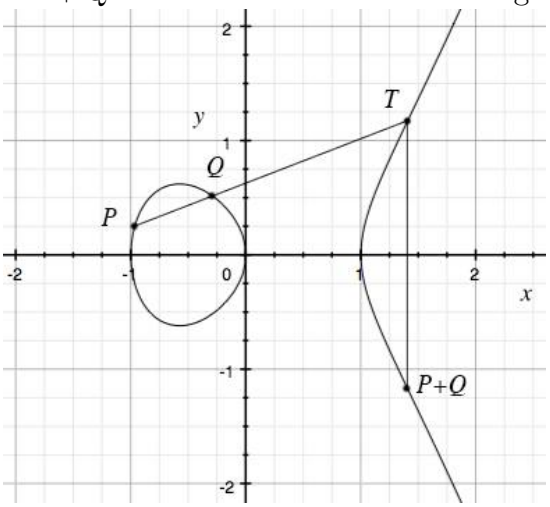

Figure 5: $P+P$

Algebraic formulas for point addition in $E(F)$ ($F=\mathbb{R}$ or $F=\mathbb{Z}_p$)

1. $P+\infty = P = \infty + P, \quad \forall P \in F.$
2. If $P=(x,y)$ and $Q=(x,-y)$ then $P+Q=\infty$. Here, we write $Q=-P$. Therefore $\infty = -\infty$.
3. If $P=(x_1,y_1)$, $Q=(x_2,y_2) \in E(F)$, $P \neq \pm Q$, then $P+Q=(x_3,y_3)$, where $\ell : y = y_1 + \lambda(x-x_1)$, $\lambda = \frac{y_2-y_1}{x_2-x_1}$. Substitute for y in $y^2 = x^3+ax+b$: $x^3+ax+b-(y_1+\lambda(x-x_1))^2 = 0 = (x-x_1)(x-x_2)(x-x_3)$.

Equate coefficients of x^2:

$$-\lambda^2 = -x_1 - x_2 - x_3$$
$$x_3 = \lambda^2 - x_1 - x_2.$$

The y-coordinate of T is $y_3 = y_1 + \lambda(x_3 - x_1)$. So the y-coordinate of R is $y_3 = -y_3'$.

$$x_3 = \lambda^2 - x_1 - x_2, \qquad \text{where } \lambda = \frac{y_2 - y_1}{x_2 - x_1}$$
$$y_3 = -y_1 + \lambda(x_1 - x_3)$$

4. If $P - (x_1, y_1)$, $P \neq -P$, then $P + P = 2P = (x_3, y_3)$ where

$$x_3 = \lambda^2 - 2x_1 \text{ where } \lambda = \frac{3x_1^2 + a}{2y_1}$$
$$y_3 = -y_1 + \lambda(x_1 - x_3)$$

Here, λ is obtained from partial derivatives of $y^2 = x_3 + ax + b$.

Addition in $E(F)$ has the following properties

1. $P + \infty = \infty + P = P, \quad \forall P \in E(F)$.
2. For each $P \in E(F), \quad \exists Q \in E(F)$ such that $P + Q = \infty$.
3. $P + Q = Q + P, \quad \forall P, Q \in E(F)$.
4. $P + (Q + R) = (P + Q) + R, \quad \forall P, Q, R \in E(F)$ (needs proof).

That is, $E(F)$ is an *Abelian group.*

Elliptic curve discrete log problem (ECDLP)

E = elliptic curve over $\mathbb{Z}_p$, $q = \#E(\mathbb{Z}_p), P \neq \infty$, and suppose q is prime. Then $\{P, 2P, 3P, \ldots, qP\} = E(\mathbb{Z}_p)$.

ECDLP:
Given p, E, q, P and $Q \in E(\mathbb{Z}_p)$, find $\ell \in [0, q-1]$ such that $\ell P = Q$.
Write $\ell = \log_P Q$.

Elliptic Curves Continued March 21

E.g. $E : y^2 = x^3 + 2x + 9$ over $\mathbb{Z}_{13}$. We have $q = 17$. Let $P = (0, 3)$.

$$\begin{aligned} P &= (0,3) & 10P &= (4,9) \\ 2P &= (3,9) & 11P &= (8,11) \\ 3P &= (1,8) & 12P &= (6,4) \\ 4P &= (11,7) & 13P &= (11,6) \\ 5P &= (6,9) & 14P &= (1,5) \\ 6P &= (8,2) & 15P &= (3,4) \\ 7P &= (4,4) & 16P &= (0,10) \\ 8P &= (5,12) & 17P &= \infty \\ 9P &= (5,1) \end{aligned}$$

Input size: $\mathcal{O}(logp)$.

- Fastest known method for solving ECDLP has runtime

  $$\mathcal{O}(\sqrt{q}) = \mathcal{O}(p^{1/2})$$

  which is fully exponential in $\log_2 p$.
- No *subexponential* algorithm is known (in general) for ECDLP.

Elliptic Curve Diffie-Hellman (ECDH):

Domain Parameters:

p, E, q, P.

Alice	Communications	Bob
$x \in_R [1, q-1]$	$X = xP \rightarrow$	
	$\leftarrow Y = yP$	$y \in_R [1, q-1]$
$K = xY$		$K = yX$

RSA vs. DL vs ECC Key Size Comparison

Security Level	Block Cipher	Hash Function	RSA Bitlength of n	DL Bitlength of p	ECC Bitlength
80	SPIPJACK	SHA-1	1024	1024	160
112	Triple-DES	SHA-224	2048	2048	224
128	AES-small	SHA-256	3072	3072	256
192	AES-medium	SHA-384	7680	7680	384
256	AES-large	SHA-512	15360	15360	512

TL;DR: ECC fast. RSA not so fast.

ECDSA:

Domain parameters: p, E, q, P.

Key generation:

Alice does:

1. Select $x \in_R [1, q-1]$.
2. Compute $X = xP$.
3. Public key: X. Private key: x.

Signature generation:

To sign $M \in \{0,1\}^*$, Alice does:

1. Compute $m = H(M)$.
2. Select $k \in_R [1, q-1]$.
3. Compute $R = kP$ and $r = x(R) \pmod q$.
4. Compute $x = k^{-1}(m + xr) \pmod q$.
5. Alice's signature on M is (r, s).

Signature verification:

To verify $M, (r, s)$, Bob does:

1. Obtain an authentic copy of Alice's public key X.
2. Verify that $1 \leq r, s \leq q-1$.
3. Compute $m = H(M)$.
4. Compute $u_1 = s^{-1}mP$ and $u_2 = s^{-1}rX$.
5. Compute $v = x(u_1 + u_2) \pmod q$.
6. Accept iff $v = r$.

‘

2005 NSA published "SUITE B".

	Secret	Top Secret
Encryption	AES-128	AES-256
Hashing	SHA-256	SHA-384
Key Agreement	ECDH-P256	ECDH-384
Signing	ECDSA-P256	ECDSA-384

Generic Construction of a PRBG (Pseudo Random Bit Generator) March 26

Ingredients

- D =finite set (e.g. $\{0,1\}^{128}$)
- Bijection $f : D \to D$, which is efficiently computable
- Function $B : D \to \{0,1\}$ such that

  (*i*) Given $x \in_R D$, it is computationally infeasible to compute $B(x)$ with probability significantly greater than $1/2$.

  (*ii*) However, given the preimage $y = f^{-1}(x)$, then computing $B(x)$ is easy. ($\Rightarrow$ finding preimages of f is hard.)

Generator G

- $x_0 \in_R D$ (the seed)
- Apply f:

$$\begin{array}{lll} x_0 \xrightarrow{f} x_1 \xrightarrow{f} & x_2 \xrightarrow{f} \ldots \xrightarrow{f} & x_{m-1} \xrightarrow{f} x_m \\ b_{m-1} & b_{m-2} \ldots & b_1 b_0 \end{array}$$

  Output: $b_0, b_1, \ldots b_{m-1}$.

- **Theorem:** G is cryptographically strong.

  Proof. Assume G is not cryptographically strong. By Yao's Theorem, we know that G fails the next bit test. So we have a prediction algorithm $\mathcal{A}$, which, when given the first ℓ bits ($1 \leq \ell \leq m-1$) of an output equence of G, efficiently computes the $(\ell+1)^{\text{st}}$ bit with probability significantly greater than $1/2$.

  Suppose we are given $x \in_R D$. We'll use $\mathcal{A}$ to compute $B(x)$ efficiently. Compute:

  So $b_0, b_1, \ldots b_{\ell-1}$ are the first ℓ bits of an output sequence of G. Hence $\mathcal{A}$ can efficiently predict $b_\ell = B(x_{m-\ell}) = B(x)$ with probability significantly greater than $1/2$.

  This contradicts the definition of B. Therefore G is cryptographically strong. □

Next time we will give an example of one such B using DLOG.