MATH 158
FINAL EXAM
17 DECEMBER 2015

Name: Solutions

- The time limit is three hours.
- No calculators are permitted.
- You are permitted one page of notes, front and back.
- The textbook's summary tables for the systems we have studied are provided at the back.
- For any problem asking you to write a program, you may write in a language of your choice or in pseudocode, as long as your answer is sufficiently specific to tell the runtime of the program.
- Point values are as indicated in the table below.

<table>
<thead>
<tr>
<th></th>
<th>/10</th>
<th>2</th>
<th>/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>/10</td>
<td>4</td>
<td>/10</td>
</tr>
<tr>
<td>5</td>
<td>/10</td>
<td>6</td>
<td>/10</td>
</tr>
<tr>
<td>7</td>
<td>/10</td>
<td>8</td>
<td>/10</td>
</tr>
<tr>
<td>9</td>
<td>/10</td>
<td>10</td>
<td>/10</td>
</tr>
<tr>
<td>11</td>
<td>/15</td>
<td>12</td>
<td>/15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/130</td>
</tr>
</tbody>
</table>
(1) Consider the elliptic curve $Y^2 = X^3 + X - 1$ over $\mathbb{Z}/5$.
(a) Determine the number of points on this curve (including the point $\mathcal{O}$).

**Squares modulo 5:** 
- $0^2 \equiv 1, 1^2 \equiv 1, 2^2 \equiv 4, 3^2 \equiv 4, 4^2 \equiv 1$, i.e. 0, 1, and 2.

**Values of $x^3 + x - 1$:**
- $0^2 + 0 - 1 \equiv 4 \Rightarrow 2$ ph w/ $x = 0$
- $1^3 + 1 - 1 \equiv 1 \Rightarrow 2$ ph w/ $x = 1$
- $2^3 + 2 - 1 \equiv 4 \Rightarrow 2$ ph w/ $x = 2$
- $3^3 + 3 - 1 \equiv 4 \Rightarrow 2$ ph w/ $x = 3$
- $4^3 + 4 - 1 \equiv 2 \Rightarrow 0$ ph w/ $x = 4$.

So there are $2 + 2 + 2 = 8$ finite points plus $\mathcal{O}$.

So **total 9 points** in all.

(b) Determine the order of the point $P = (1, 1)$.

By (a), $\text{ord}(P) \mid 9$, so the order is 1, 3, or 9. It can't be 1 since $P \neq \mathcal{O}$.

We need only check whether or not $3P = \mathcal{O}$.

$2P = (1,1) \oplus (1,1)$
- $\lambda = (3 \cdot 1^2 + 1) \cdot (2 \cdot 1)^{-1} \equiv 4 \cdot 2^{-1} \equiv 2 \mod 5$
- $x = \lambda^2 - 1 - 1 = 2$
- $y = -(1 + 2 \cdot (2 - 1)) \equiv -3 \equiv 2 \mod 5$

$\Rightarrow 2P = (2, 2)$

$3P = (2, 2) \oplus (1,1)$
- $\lambda = (2 \cdot 2 - 1) \cdot (2^{-1}) \equiv 1 \mod 5$
- $x = \lambda^2 - 2 - 2 = 1$

Since $(2, 2) \neq \mathcal{O} (1, 1)$, $3P \neq \mathcal{O}$. (no need to compute it).

So $\text{ord} P \neq 1, 3$; it follows that $\text{ord} P = 9$. 
(2) Explain briefly why each of the following choices is made in DSA. Be specific about which mathematical facts would make the algorithm either incorrect or insecure otherwise.

(a) The number \( q \) is a prime number.

- \( q \) is the order of \( q \), & the sig. scheme is insecure if Eve can do discrete logarithms base \( q \) modulo \( p \).
- If \( q \) is composite, Eve can use Pohlig-Hellman to reduce her effort to instances of DLP when the order of the base is a factor of \( q \); these would be much easier.
- A prime ensures this attack is not useful.

(b) The numbers \( p, q \) satisfy \( p \equiv 1 \pmod{q} \).

\( \mathbb{Z}/p)^* \) has an element of order \( q \) if and only if \( q \mid (p-1) \), i.e. \( p \equiv 1 \pmod{q} \).

Thus finding \( q \) would be impossible & the algorithm wouldn't work otherwise.

(c) The number \( k \) is selected at random.

If Eve learns \( k \), or if Eve finds that a value of \( k \) is ever used twice, she can learn the secret signing key from a signature. Choosing a new \( k \) at random each time eliminates this risk, making the system more secure.
(3) Alice's RSA public key has modulus \( N \). Bob cannot remember whether her encrypting exponent is 16 or 27. In a well-meaning but very foolish blunder, he decides to encrypt his message \( m \) with both possible encrypting exponents, creating \( c_1 \) (using \( e = 16 \)) and \( c_2 \) (using \( e = 27 \)). Bob uses the correct modulus \( N \) in both cases. He then sends both \( c_1 \) and \( c_2 \) to Alice, with an explanation of what happened.

Eve intercepts \( c_1 \) and \( c_2 \), as well as the information of which exponent was used to create which ciphertext. Show that \( m \) can be expressed in terms of \( c_1 \) and \( c_2 \) using arithmetic modulo \( N \) and hence that Eve can learn the plaintext \( m \).

\[
\begin{align*}
C_1 &\equiv m^{16} \mod N \\
C_2 &\equiv m^{27} \mod N \\
\implies C_1^u C_2^v &\equiv m^{16u + 27v} \mod N \quad \forall u, v \in \mathbb{Z}.
\end{align*}
\]

We can use ext. Euclid to express 1 as \( 16u + 27v \):

\[
\begin{align*}
27 &= 1 \cdot 27 \\
16 &= 1 \cdot 16 \\
11 &= -1 \cdot 16 + 1 \cdot 27 \\
5 &= 2 \cdot 16 - 1 \cdot 27 \\
1 &= 4 \cdot 16 - 5 \cdot 16 + 3 \cdot 27
\end{align*}
\]

So

\[
m \equiv C_1^{-5} \cdot C_2^3 \mod N.
\]

Inverses \( \mod N \) \& powers \( \mod N \) are both \( \text{efficient} \), so Eve can compute \( m \) this way very easily.
(4) The following function definition is meant to calculate the sum of two points $P, Q$ on the elliptic curve $Y^2 = X^3 + AX + B$ over $\mathbb{Z}/p$, but it contains a flaw. Explain the case in which the code will not work properly, and how to fix it.

**Assumptions:** each point $(P, Q$, or the return value) is either a pair $(x, y)$ of two integers with $0 \leq x, y < p$, or the number 0 (for the point $O$). You may assume that both $P$ and $Q$ do in fact lie on the curve defined by $A$ and $B$. Also assume that $\text{inv}\_\text{mod}(a, m)$ is a correctly implemented function that returns the inverse of $a$ modulo $m$ whenever $a$ is a unit modulo $m$, but which results in an error if $a$ is not a unit modulo $m$.

```python
def add(P, Q, A, B, p):
    if P == 0: return Q
    if Q == 0: return P
    if P[0] == Q[0] and P[1] != Q[1]: return 0
    if P[0] != Q[0]:
        rise = (P[1] - Q[1]) % p
        run = (P[0] - Q[0]) % p
    else:
        rise = (3*P[0]*P[0] + A) % p
        run = (2*P[1]) % p
    slope = (rise*\text{inv}\_\text{mod}(run, p)) % p
    y_int = (P[1] - P[0]*slope) % p
    x = (slope*slope - P[0] - Q[0]) % p
    y = (-slope*x + y_int) % p
    return (x, y)
```

The marked line is meant to detect the case $P = OQ$ and return $O$ in this case. However, if $P = (x, 0)$ where $x^3 + Ax + B \equiv 0 \mod p$ and $Q = (x, 0)$ then it will fail; later an error will occur since “run” will be 0 & have no inverse.

An easy fix is to replace this line with:

```python
    iF P[0] == Q[0] and (P[1] + Q[1]) % p == 0: return 0
```
(5) Write a function `pickg(p, q)` with the following behavior: if \( p, q \) are both prime numbers, then the return value must be either a number \( a \) between 1 and \( p - 1 \) inclusive with order \( q \) modulo \( p \), or the number \(-1\) if no such integer \( a \) exists. Your function may be randomized. For full points the (expected value of the) number of arithmetic operations performed by the function must be \( O(\log p) \).

```python
import random
def pickg(p, q):
    if (p-1)%q != 0: return -1
    while True:
        a = random.randrange(2, p)
        g = pow(a, (p-1)/a, p)
        if g != 1: return g
```
(6) Suppose that Samantha is using ECDSA parameters with \( q = 7 \). She has published two valid signatures: \( (2, 3) \) for the document \( d = 4 \), and \( (2, 6) \) for the document \( d' = 5 \). Eve learns that she used the same random element \( e \) to produce both signatures. Determine Samantha’s secret signing key, \( s \).

\[
S_2 \equiv (d + s \cdot s_1) e^{-1} \mod q
\]
i.e.
\[
e \cdot S_2 - s \cdot s_1 \equiv d \mod q
\]
for any valid signature.

For these two:

\[
e \cdot 3 - s \cdot 2 \equiv 4 \mod 7
\]
\[
e \cdot 6 - s \cdot 2 \equiv 5 \mod 7
\]

Subtracting the first from the second twice:

\[
e \cdot (6-2 \cdot 3) + s (-2+2 \cdot 2) \equiv 5-2 \cdot 4
\]
\[\Rightarrow\]
\[
2s \equiv -3 \equiv 4 \mod 7
\]
\[\Rightarrow\]
\[
S \equiv 2 \mod 7
\]

One example of what the full information could be:

\[
\text{curve: } Y^2 = X^3 + 2X + 4 \text{ over } \mathbb{F}_{15}
\]
\[
G = (4, 1) \text{ is a pt. of ord } q = 7
\]

\[
\text{secret key } s = 2 \text{ varl. key } V = (2, 4) \text{ (= 2, 6)}.
\]
\[
e = 5 \text{ for both signatures.}
\]
(7) Suppose that Eve has intercepted a ciphertext from Bob to Alice. In addition, she knows by other means that the plaintext is one of only 1000 possibilities (for example, it might specify a landmark where Alice and Bob will meet, written in a predictable format and chosen from a short list of options). As usual, Eve knows Alice’s public key, but not her private key.

(a) Suppose that the cryptosystem being used is RSA. Explain how Eve can very quickly identify for certain which of the 1000 candidates is the true plaintext.

For each candidate \( m_1, m_2, \ldots, m_{1000} \),
Eve just encrypts it, computing
\[
C_i \equiv m_i^e \mod N \quad \text{[\((N,e) \text{ is the public key}\]}]
\]

Exactly one \( C_i \) will be the intercepted ciphertext.

\textbf{Encryption is one-to-one, so Eve knows } m_i \text{ is the plaintext.}

(b) Suppose that the cryptosystem being used is Menezes-Vanstone (table 6.13). Describe a procedure Eve could use that, \textit{with very high probability}, will pick out the correct plaintext from the list. (More formally: your procedure should have the property that if the 999 false plaintexts were chosen uniformly at random, then the probability of choosing one of them should be negligible.)

Let the candidates be \((m_{i1}, m_{i2})\) for \(i = 1, 2, \ldots, 1000\), and the ciphertext be \((C_i, C_i)\).

For each candidate, compute
\[
x_i \equiv C_i \cdot m_{i1}^{-1} \mod p
\]
\[
y_i \equiv C_i \cdot m_{i2}^{-1} \mod p.
\]

If \((x_i, y_i)\) doesn’t lie on the curve, the candidate can be thrown out. (If correct, \((x_i, y_i) = (x_{1i}, y_{1i})\))

The odds of a false candidate not being thrown out are slim, since \(\approx \frac{1}{5}\) possible pairs \((x, y)\) actually lie on the curve. So most likely only the true plaintext will remain.
(8) The NTru procedure (table 7.4) stipulates that $p$ and $q$ should be chosen such that $\gcd(p, q) = 1$. Suppose that parameters are chosen that do not obey this rule, and instead $p \mid q$. In this case, the system is completely insecure. Write a function that Eve could use to can break it.

Specifically: write a function $\text{extract}(e, N, p, q, d, h)$ that efficiently extracts the plaintext $m$ from any cipher text $e$, given only the public key and system parameters. The arguments $e$ and $h$ will be given as lists of $N$ integers. The coefficients in your answer should be either centerlifted modulo $p$ or reduced modulo $p$ in the typical way.

\[ e \equiv p \cdot h \cdot r + m \mod q. \]

Since $p \mid q$, it follows that all coeff of both sides are congruent $\mod p$ as well as $b \mod q$.

So

\[ e \equiv p \cdot h \cdot r + m \mod p \]
\[ \equiv 0 \cdot h \cdot r + m \mod p \]
\[ \equiv m \mod p. \]

So the function is extremely simple to write.

```python
def extract(e, N, p, q, d, h):
    return [ei % p for ei in e]
```
(9) Suppose that $P, Q$ are two points on an elliptic curve over $\mathbb{Z}/9719$ (the number $p = 9719$ is prime). The order of the elliptic curve is a prime number $q$, and neither $P$ nor $Q$ is $\mathcal{O}$. Alice has constructed the following two lists of points.

\[
[\mathcal{O}, P, 2P, \ldots, 99P] \\
[Q, Q \oplus 100P, Q \oplus 200P, \ldots, Q \oplus 9900P]
\]

Prove that there must exist a common element between these two lists, and describe how finding this common element can be used to find an integer $n$ such that $Q = nP$.

Since $q$ is prime, every non-$\mathcal{O}$ point on the curve has order $q$. So $\mathcal{O}, P, 2P, \ldots, (q-1)P$ are all distinct (since $iP \equiv iP$ would imply $(i-j)P = \mathcal{O}$, i.e., $a \equiv a\mod{(i-j)}$), hence all pts. on the curve are equal to a multiple of $P$.

By Hasse's theorem,

\[
q \leq p + 1 + 2\sqrt{p} \\
\leq 9719 + 1 + 2\sqrt{9719} \\
< 9720 + 2\sqrt{10000} = 9720 + 200 \\
< 10,000.
\]

So $\{\mathcal{O}, P, 2P, \ldots, 9999P\}$ includes the point $Q$ somewhere, say at $nP$ ($n < 10,000$). Let

\[
i = n \div 100 \quad \text{so} \quad n = i + 100j
\]

Both $i, j$ are between 0 & 99 inclusive. so $iP$ lies in the first list

& $Q \oplus j\cdot100P$ lies in the second

& there are equal.

Conversely, if $iP = Q \oplus (100j)P$ for some $i, j$, then $Q = (i + 100j)P$,

so $n = i + 100j$ is the desired value.
(10) Suppose that the NTru cryosystem (Table 7.4) is modified in the following ways.

- The single integer \( d \) in the parameters is replaced with three integers \( d_1, d_2, d_3 \) such that \( d_1 > d_2 > d_3 \). The requirement that \( q > (6d + 1)p \) is removed.
- When Alice chooses \( f \), she chooses it from \( T(d_3 + 1, d_1) \).
- When Alice chooses \( g \), she chooses it from \( T(d_2, d_2) \).
- When Bob chooses \( r \), he chooses it from \( T(d_3, d_3) \).

Derive an inequality of the form “\( q > \cdots \)” (to replace \( q > (6d + 1)p \) from the original version) in terms of \( d_1, d_2, d_3 \) (not all three of which must necessarily be used) and the other public parameters, such that decryption is guaranteed to succeed as long as this inequality holds.

As before, we require that \( p \mathbb{Z}[c] + m \mathbb{Z} \) is its own centerlift modulo \( q \) in order for decryption to work. That is, all coefficients must be less than \((0, \frac{q}{2}, \frac{q}{2}) \); it suffices for them to be below \( \frac{q}{2} \) in absolute value.

A coefficient \( \mathbb{Z}[c] \) can be written as the sum of \( d_3 \) coefficients of \( g \) minus \( d_3 \) coefficients of \( g \) (since \( c \) has \( d_3 + 1 \)'s & \( d_3 - 1 \)'s);

all coefficients of \( g \) are \( \pm 1 \) or 0, so \( |g|_{\mathbb{Z}[c]} \leq 2d_3 \),

and \( |pg|_{\mathbb{Z}[c]} \leq 2pd_3 \).

A coefficient \( m \) is a sum of \( d_3 + 1 \) coefficients of \( m \) minus a sum of \( d_3 \) terms, all terms of \( m \) are \( \leq \frac{p}{2} \) in abs.

value, so \( |m| \leq (2d_3 + 1) \frac{p}{2} \).

Thus \( |pg + m| \leq 2pd_3 + (2d_3 + 1) \frac{p}{2} = (4d_3 + 2d_3 + 1) \frac{p}{2} \).

Decryption works as long as this is \( < \frac{q}{2} \). Hence it suffices to ensure that

\[
q > (2d_1 + 4d_3 + 1)p
\]

(note: if \( d_1 = d_2 = d_3 \), we recover the original inequality).
(11) Samantha and Victor agree to the following digital signature scheme. The public parameters and key creation are identical to those of ECDSA. The verification procedure is different: to decide whether \((s_1, s_2)\) is a valid signature for a document \(d\), Victor computes

\[
\begin{align*}
w_1 &\equiv s_1^{-1}d \pmod{q} \\
w_2 &\equiv s_1^{-1}s_2 \pmod{q},
\end{align*}
\]

then he check to see whether or not

\[
x(w_1G + w_2V)\%q = s_1.
\]

If so, he regards \((s_1, s_2)\) as a valid signature for \(d\).

(a) Describe a signing procedure that Samantha can follow to produce a valid signature on a given document \(d\). The procedure should be randomized in such a way that it will generate different signatures if executed repeatedly on the same document.

The equation \(x(w_1G + w_2V)\%q = s_1\) of integers is guaranteed by the equation

\[
w_1G + w_2V = e \cdot G
\]

of points on the curve,

which is equivalent to the congruence

\[
\begin{align*}
w_1 + w_2 \cdot s &\equiv e \bmod q \\
\equiv &\quad s_1^{-1}d + s_1^{-1}s_2s \\&\equiv e \bmod q \\
\equiv &\quad d + s_1s_2 \\&\equiv e \cdot s_1 \bmod q \\
\equiv &\quad s_2 \equiv s_1^{-1}(e \cdot s_1 - d) \bmod q.
\end{align*}
\]

So Samantha's signing procedure can be:

1. Choose a random \(s \in \mathbb{Z}/q\).
2. \(s_1 = x(e \cdot G) \%q\)
3. \(s_2 \equiv s_1^{-1}(e \cdot s_1 - d) \bmod q\).
(b) Describe a forgery procedure that Eve can follow to create a signature \((s_1, s_2)\) and a document \(d\) such that \((s_1, s_2)\) is a valid signature for \(d\) under this scheme. Note that Eve does not need to be able to choose \(d\) in advance. The procedure should be randomized in such a way that it can generate many different forgeries (on many different documents).

(see P. Set & #1, 1b For an analogous construction)

As in the examples from class, Eve can get a bit of flexibility by choosing two numbers \(i, j\) at random and setting \(s_1\) as follows:

\[ s_1 = x(i \cdot G \oplus j \cdot V) \equiv q \]

(thus, if \(P\) Samantha chose \(e\) to be \(i + j\), but
Eve can do it without knowing \(e\)).

Then she must choose \(s_2\) and \(d\) such that:

\[ i \cdot G \oplus j \cdot V = w \cdot G \oplus w_j \cdot V \]

\[ \Leftrightarrow \quad i + j a \equiv s_1 \cdot d + s_i \cdot s_2 \cdot a \mod q \]

\[ \Leftrightarrow \quad s_i \cdot s_2 \cdot a \equiv d + s_2 \cdot a \mod q \]

\[ \Leftrightarrow \quad (s_i \cdot s_2 - d) \equiv (s_i - s_2) \cdot a \mod q. \]

Eve can eliminate the need to know \(a\) by setting

\[ s_2 \equiv s_i \cdot j \mod q \]

\[ d \equiv s_i \cdot i \mod q, \]

which will satisfy the desired congruence.

In summary:

1) choose \(i, j \in \mathbb{Z}/q\) at random.
2) set (in order):

\[ s_1 = x(i \cdot G \oplus j \cdot V) \equiv q \]

\[ s_2 \equiv s_i \cdot j \mod q \]

\[ d \equiv s_i \cdot i \mod q, \]
(12) Suppose that $n$ is an odd integer such that exactly $\frac{1}{32}$ of all units modulo $n$ are squares (i.e. are congruent to some integer square modulo $n$). Alice wishes to factor $n$. Suppose that Alice chooses $m$ distinct elements $a_1, a_2, \ldots, a_m$ of $\{1, 2, \ldots, \frac{n-1}{2}\}$ at random.

(a) Suppose that Alice discovers that $a_i^2 \equiv a_j^2 \pmod{n}$ for some $i \neq j$. Write a function $\text{factor}(n, a_i, a_j)$ which returns a proper factor (i.e. a factor besides 1 or $n$) of $n$ given the values $a_i$ and $a_j$ whose squares are congruent. For full credit, your function should perform no more than $O(\log n)$ arithmetic operations.

Note that $a_i^2 \equiv a_j^2 \pmod{n}$ means that $\gcd(a_i + a_j, a_i - a_j)$. Since $0 < a_i + a_j \leq n-1$, $n \nmid (a_i + a_j)$. Similarly, $-n < a_i - a_j < n$ & $a_i - a_j \neq 0$, so $n \nmid (a_i - a_j)$. So $n$ divides the product, some factor must divide $a_i + a_j$, some divide $a_i - a_j$, but $n$ divides neither. So $\gcd(n, a_i + a_j)$ and $\gcd(n, a_i - a_j)$ are both proper factors. We can find either one with the Euclidean algorithm.

def factor(n, ai, aj):
    # finds gcd(n, ai + aj).
    a, b = n, ai + aj
    while b != 0:
        a, b = b, a % b
    return a
(b) Assuming that all \( m \) of these elements are (distinct) units modulo \( n \), prove that the probability that \( a^2 \equiv a_j^2 \pmod{n} \) for some \( i \neq j \) is at least \( e^{-\text{as}(\varphi(n)/\varphi(n))} \). You may assume without proof that \( e^{-x} \leq 1 - x \) for all real numbers \( x \).

There are \( \varphi(n) \) different squares to choose from \( (\varphi(n) \) units of which \( 1/2 \) are squares). So the prob. of choosing \( n \) different values of \( a^2 \) mod \( n \) is:

\[
1 \cdot \left( 1 - \frac{1}{\varphi(n)/12} \right) \cdot \left( 1 - \frac{2}{\varphi(n)/12} \right) \cdots \left( 1 - \frac{m-1}{\varphi(n)/12} \right)
\]

\[
= \frac{\varphi(n)}{\varphi(n)} \cdot \left( 1 - \frac{1}{\varphi(n)/12} \right) \cdot \left( 1 - \frac{2}{\varphi(n)/12} \right) \cdots \left( 1 - \frac{m-1}{\varphi(n)/12} \right)
\]

\[
\leq e^0 \cdot e^{-32/\varphi(n)} \cdot e^{-2.72/\varphi(n)} \cdots e^{-(m-1) \cdot 32/\varphi(n)}
\]

\[
= e^{-32(0+1+2+\cdots+(m-1))/\varphi(n)} = e^{-32(m^2)/\varphi(n)}
\]

So the prob. that there is some collision is \( e^{-32(m^2)/\varphi(n)} \) as desired.

(c) Suppose that the assumption in part (b) fails, and in fact one of the \( a_i \) is \textbf{not} a unit modulo \( n \). This is a feature, not a bug: describe how Alice can quickly find a proper factor of \( n \) in this case, before she even looks for any collisions.

\( a_i \) is a nonunit \( \implies \gcd(a_i, n) \neq 1 \).

Since \( a_i < n \), \( \gcd(a_i, n) \) is a proper factor of \( n \); we \( \tau \) can just return it now and stop looking for more \( a_i \).
“Bonus” (to keep me happy during grading, not for real points): fill in cryptography-related (or totally unrelated) dialog for this comic.

I've got to tell Alice...

A SECRET

Hi Bob! Where are you headed today?

Eve! Nowhere! I was just leaving!

Wait up, Bob! I thought you had news?

No! Eve will hear it!

Why not just use my public key?

You're right! I'll go encrypt my message!

Later...

Modular arithmetic is hard!
Public parameter creation
A trusted party chooses and publishes a large prime \( p \) and an integer \( g \) having large prime order in \( \mathbb{Z}_p^* \).

Private computations

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose a secret integer ( a ).</td>
<td>Choose a secret integer ( b ).</td>
</tr>
<tr>
<td>Compute ( A = g^a \pmod{p} ).</td>
<td>Compute ( B = g^b \pmod{p} ).</td>
</tr>
</tbody>
</table>

Public exchange of values

Alice sends \( A \) to Bob, \( B \) sends \( B \) to Alice.

Further private computations

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute the number ( B^a \pmod{p} ).</td>
<td>Compute the number ( A^b \pmod{p} ).</td>
</tr>
<tr>
<td>The shared secret value is ( B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p} ).</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Diffie-Hellman key exchange

Public parameter creation
A trusted party chooses and publishes a large prime \( p \) and an element \( g \) modulo \( p \) of large (prime) order.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key creation</td>
<td></td>
</tr>
<tr>
<td>Choose private key ( 1 \leq a \leq p-1 ).</td>
<td></td>
</tr>
<tr>
<td>Compute ( A = g^a \pmod{p} ).</td>
<td></td>
</tr>
<tr>
<td>Publish the public key ( A ).</td>
<td></td>
</tr>
</tbody>
</table>

Encryption

Choose plaintext \( m \).
Choose random element \( k \).
Use Alice’s public key \( A \) to compute \( c_1 = g^k \pmod{p} \) and \( c_2 = mA^k \pmod{p} \).
Send cipher text \( (c_1, c_2) \) to Alice.

Decryption

\( \left(c_1^a\right)^{-1} \cdot c_2 \pmod{p} \).
This quantity is equal to \( m \).

Table 2.3: ElGamal key creation, encryption, and decryption

<table>
<thead>
<tr>
<th>Bob</th>
<th>Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key creation</td>
<td></td>
</tr>
<tr>
<td>Choose secret primes ( p ) and ( q ).</td>
<td></td>
</tr>
<tr>
<td>Choose encryption exponent ( e ) with ( \gcd(e, p-1)(q-1) = 1 ).</td>
<td></td>
</tr>
<tr>
<td>Publish ( N = pq ) and ( e ).</td>
<td></td>
</tr>
</tbody>
</table>

Encryption

Choose plaintext \( m \).
Use Bob’s public key \( (N, e) \) to compute \( c \equiv m^e \pmod{N} \).
Send ciphertext \( c \) to Bob.

Decryption

\( ed \equiv 1 \pmod{(p-1)(q-1)} \).
Compute \( m' \equiv c^d \pmod{N} \).
Then \( m' \) equals the plaintext \( m \).

Table 3.1: RSA key creation, encryption, and decryption

Public parameter creation
A trusted party chooses and publishes large primes \( p \) and \( q \) satisfying \( p \equiv 1 \pmod{q} \) and an element \( g \) of order \( q \) modulo \( p \).

<table>
<thead>
<tr>
<th>Samantha</th>
<th>Victor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key creation</td>
<td></td>
</tr>
<tr>
<td>Choose secret primes ( p ) and ( q ).</td>
<td></td>
</tr>
<tr>
<td>Choose verification exponent ( e ) with ( \gcd(e, (p-1)(q-1)) = 1 ).</td>
<td></td>
</tr>
<tr>
<td>Publish ( N = pq ) and ( e ).</td>
<td></td>
</tr>
</tbody>
</table>

Signning

Choose \( d \) satisfying \( de \equiv 1 \pmod{(p-1)(q-1)} \).
Sign document \( D \) by computing \( S \equiv D^e \pmod{N} \).

Verification

Compute \( S^a \pmod{N} \) and verify that it is equal to \( D \).

Table 4.1: RSA digital signatures

Public parameter creation
A trusted party chooses and publishes large primes \( p \) and \( q \) satisfying \( p \equiv 1 \pmod{q} \) and an element \( g \) of order \( q \) modulo \( p \).

<table>
<thead>
<tr>
<th>Samantha</th>
<th>Victor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key creation</td>
<td></td>
</tr>
<tr>
<td>Choose secret signing key ( 1 \leq a \leq p-1 ).</td>
<td></td>
</tr>
<tr>
<td>Compute ( A = g^a \pmod{p} ).</td>
<td></td>
</tr>
<tr>
<td>Publish the verification key ( A ).</td>
<td></td>
</tr>
</tbody>
</table>

Signning

Choose random element \( 1 < k < p \) satisfying \( \gcd(k, p-1) = 1 \).
Compute signature \( S_1 \equiv g^k \pmod{p} \) and \( S_2 \equiv (D-aS_1)^{k-1} \pmod{p-1} \).

Verification

Compute \( A^{S_1}S_2^{S_2} \pmod{p} \).
Verify that it is equal to \( g^D \pmod{p} \).

Table 4.2: The ElGamal digital signature algorithm

Public parameter creation
A trusted party chooses and publishes large primes \( p \) and \( q \) satisfying \( p \equiv 1 \pmod{q} \) and an element \( g \) of order \( q \) modulo \( p \).

<table>
<thead>
<tr>
<th>Samantha</th>
<th>Victor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key creation</td>
<td></td>
</tr>
<tr>
<td>Choose secret signing key ( 1 \leq a \leq q-1 ).</td>
<td></td>
</tr>
<tr>
<td>Compute ( A = g^a \pmod{p} ).</td>
<td></td>
</tr>
<tr>
<td>Publish the verification key ( A ).</td>
<td></td>
</tr>
</tbody>
</table>

Signning

Choose random element \( 1 < k < q \).
Compute signature \( S_1 \equiv (g^a \pmod{p})^{m} \pmod{q} \) and \( S_2 \equiv (D-aS_1)^{k-1} \pmod{q} \).

Verification

Compute \( V_1 \equiv D^{S_1}S_2^{-1} \pmod{q} \) and \( V_2 \equiv S_1^{-1} \pmod{q} \).
Verify that \( (g^{y_1}A^{y_2} \pmod{p}) \pmod{q} = S_1 \).

Table 4.3: The digital signature algorithm (DSA)
Table 6.5: Diffie–Hellman key exchange using elliptic curves

<table>
<thead>
<tr>
<th>Public parameter creation</th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses a finite field ( \mathbb{F}_p ), an elliptic curve ( E/\mathbb{F}_p ), and a point ( O \in E(\mathbb{F}_p) ) of large prime order ( q ).</td>
<td>Choose a secret integer ( n_A ).</td>
<td>Choose a secret integer ( n_B ).</td>
</tr>
<tr>
<td>Computes the point ( Q_A = n_A P ).</td>
<td>Computes the point ( Q_B = n_B P ).</td>
<td></td>
</tr>
<tr>
<td>Alice sends ( Q_A ) to Bob</td>
<td>Bob sends ( Q_B ) to Alice</td>
<td></td>
</tr>
</tbody>
</table>

Further private computations

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computes the point ( n_A Q_B ).</td>
<td>Computes the point ( n_B Q_A ).</td>
</tr>
</tbody>
</table>

The shared secret value is \( n_A Q_B = n_A (n_B P) = n_B (n_A P) = n_B Q_A \).

Table 6.7: The elliptic curve digital signature algorithm (ECDSA)

<table>
<thead>
<tr>
<th>Public Parameter Creation</th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses and publishes a (large) prime ( p ), an elliptic curve ( E ) over ( \mathbb{F}_p ), and a point ( P ) in ( E(\mathbb{F}_p) ).</td>
<td>Choose a secret multiplier ( n_A ).</td>
<td>Computes ( Q_A = n_A P ).</td>
</tr>
<tr>
<td>Publishes the public key ( Q_A ).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key Creation

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose secret signing key ( 1 &lt; s &lt; q - 1 ).</td>
<td>Choose ( s ) as a random number ( s \bmod q ).</td>
</tr>
<tr>
<td>( V = sG \in E(\mathbb{F}_p) ).</td>
<td>( \sigma = (sG) \bmod q ).</td>
</tr>
<tr>
<td>Publish the verification key ( V ).</td>
<td>Publish the signature ( s_1, s_2 ).</td>
</tr>
</tbody>
</table>

Signing

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose document ( d \bmod q ).</td>
<td>Choose random element ( e \bmod q ).</td>
</tr>
<tr>
<td>Choose ( \sigma = (d + sn) \bmod q ).</td>
<td>( \sigma = s^{-1} \bmod q ).</td>
</tr>
<tr>
<td>Compute ( v_1 \equiv s_2^{-1} \bmod q ).</td>
<td>Compute ( v_2 \equiv s_1 \bmod q ).</td>
</tr>
<tr>
<td>Publish the signature ( (s_1, s_2) ).</td>
<td>Publish the signature ( (v_1, v_2) ).</td>
</tr>
</tbody>
</table>

Verification

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computes ( v_1 \equiv ds_2^{-1} \bmod q ) and ( v_2 \equiv s_1 s_2^{-1} \bmod q ).</td>
<td>Computes ( v_1 G + v_2 V \in E(\mathbb{F}_p) ) and verify that ( v_1 G + v_2 V \equiv (d + sn) G \bmod q ).</td>
</tr>
</tbody>
</table>

Table 7.1: A congruent public key cryptosystem

<table>
<thead>
<tr>
<th>Public parameter creation</th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses public parameters ( (N, p, q, d) ) with ( N ) and ( p ) prime, ( \gcd(p, q) = \gcd(N, q - 1) = 1 ), and ( q &gt; (6d + 1)p ).</td>
<td>Choose private ( e \in T(d + 1, d) ).</td>
<td>Choose ( f \in T(d, d) ).</td>
</tr>
<tr>
<td>( F_q ), the inverse of ( f ) in ( R_q ).</td>
<td>Compute ( F_p ), the inverse of ( f ) in ( R_p ).</td>
<td></td>
</tr>
<tr>
<td>Publish the public key ( h = F_p ).</td>
<td>Publish the public key ( h = f ).</td>
<td></td>
</tr>
</tbody>
</table>

Encryption

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose plaintext ( m \in R_q ).</td>
<td>Choose ( s ) as a random number ( s \in T(d, d) ).</td>
</tr>
<tr>
<td>Use Alice's public key ( (h, q) ).</td>
<td>Use Alice's public key ( (h, q) ).</td>
</tr>
<tr>
<td>Compute ( e \equiv pr \times h + m \bmod q ).</td>
<td>Compute ( e \equiv pr \times h + m \bmod q ).</td>
</tr>
</tbody>
</table>

Decryption

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute ( f \times e \equiv pg \times r = f \times m \bmod q ).</td>
<td>Compute ( f \times e \equiv pg \times r = f \times m \bmod q ).</td>
</tr>
<tr>
<td>Center-lift to ( a \in R ) and compute ( m \equiv F_p \times a \bmod p ).</td>
<td>Send ciphertext ( e ) to Alice.</td>
</tr>
</tbody>
</table>

Table 7.4: NTRU: the NTRU public key cryptosystem

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computes ( T = n_AR ) and writes it as ( T = (2T_1, 2T_2) ).</td>
<td>Choose plaintext ( m \in R_q ).</td>
</tr>
<tr>
<td>Sets ( m_1' \equiv T_1c_1 \bmod p ) and ( m_2' \equiv T_2c_2 \bmod p ).</td>
<td>Use Alice's public key ( (h, q) ).</td>
</tr>
<tr>
<td>Then ( m_1' = m_1 ) and ( m_2' = m_2 ).</td>
<td>Compute ( e \equiv pr \times h + m \bmod q ).</td>
</tr>
</tbody>
</table>

Table 6.13: Menezes–Vanstone variant of Elgamal (Exercises 6.17, 6.18)