1. Security Scheme Definitions

Security schemes are normally presented as a game, and cryptographic systems are tested in these games to see if they are secure if an adversary cannot win a disporportionate amount of the time it.

1.1. IND-CCA (Indistinguishability under Chosen Ciphertext Attack).

Phase I:

- Examiner produces $(PK, SK) \leftarrow Keygen(1^\lambda)$.
- Adversary is given PK.
- Adversary computes in time $poly(\lambda)$ with access to decryption oracle $Dec(SK, \cdot)$ and encryption oracle $Enc(PK, \cdot)$. Adversary outputs m_0, m_1 where $|m_0| = |m_1|$. The adversary can also store state information s and obtain this information in the next phase.

Phase II:

- Examiner chooses $b \leftarrow \{0, 1\}$ and computes $y = Enc(SK, m_b)$.
- Adversary is given access to state information s and allowed to compute in time $poly(\lambda)$. Then, he produces a guess $\hat{b}$.

If adversary's advantage, defined as $|P(\hat{b} = b) - \frac{1}{2}|$, is negligible then the encryption scheme is deemed secure.

Note: Encryption must be randomized, and random values cannot be easily observable for IND-CCA security.

1.2. IND-CCA2 (Indistinguishability under Adaptive Chosen Ciphertext Attack).

Adapativity is a stronger security claim than IND-CCA. Everything in IND-CCA2 is the same, except that in phase II, the adversary is given access to the decryption block $Dec(SK, \cdot)$ on all inputs except for y.

1.3. IND-CPA (Indistinguishability under Chosen Plaintext Attack).

Almost the same as IND-CCA, but it is a little weaker. Under IND-CPA, the adversary is given access only to the encryption block, and never to the decryption block. Thus, the adversary can compute any encryptions in $poly(\lambda)$, but cannot use $Dec(SK, \cdot)$ for either Phase I or II.

Again, this security scheme requires that encryption is randomized.

1.4. Semantic Security.

Semantic Security is equivalent to IND-CPA.

This means that if you want to show IND-CPA, then it is sufficient to show semantic security, and if you want to show semantic security, it is sufficient to show IND-CPA.

Definition: A cryptosystem is *semantically secure* if any probabilistic polynomial time algorithm that is given the cipher c of message m and given $|m|$, cannot determine any other information about the message with non-neglible probability. In other words, it must be infeasible for a computationally bounded adversary to obtain significant information about a plaintext from a ciphertext and the public encryption key.

Note that semantic security does not consider CCA where the attacker is able to request the decryption of chosen ciphertexts.

Examples of semantically secure algorithms:

- Goldwasser Micali
- El Gamal
- Paillier

2. Commitment Schemes

2.1. Pedersen Commitment Scheme.

3. Cryptographic Systems

3.1. ElGamal Encryption.

ElGamal is a public key encryption scheme. The system revolves around a cyclic group $G = \langle g \rangle$ with generator g.

Key Generation: We pick x uniformly at random from $[0, \ldots, |G| - 1]$ and set $SK = x$ and $PK = g^x$.

Encryption: Pick $k \leftarrow [0, \ldots, |G| - 1]$ uniformly at random and assume m can be represented as an element of G. Now let $y = g^x$ be the public key of the recipient, we send $c = (g^k, my^k)$ as the ciphertext.

Decryption: We let $c = (a, b)$ be the received ciphertext. Compute and output ba^{-x} as m.

Proof of Correctness: We just need to show that $ba^{-x} = m$. But, we know that $b = my^k$ and $a = g^k$, which implies that $ba^{-x} = my^k(g^k)^{-x} = my^k(g^x)^{-k} = my^k y^{-k} = m$. This proves correctness.

We see that ElGamal encryption is related to the Diffie-Hellman key exchange because you encrypt by multiplying by a DH key and decrypt by dividing by a DH key.

Also we should note some facts about ElGamal encryption. First, ElGamal is *malleable*. Given some encryption $E(m) = (g^k, my^k)$ we can get a new encryption by taking some constant c and creating $E(m') = (g^k, cmy^k) = (g^k, m'y^k$ where $m' = cm$. More generally, ElGamal is *homomorphic*, i.e. multiplying two ciphertexts will create a new ciphertext which is the multiplication of the original plaintexts. Suppose we're given $c_1 = (g^r, m_1y^r)$ and $c_2 = (g^s, m_2y^s)$, then we have $c_1c_2 = (g^{r+s}, (m_1m_2)y^{r+s}) = Enc(m_1, m_2)$.

You can re-randomize the encryption by multiplying by $Enc(g^s, y^s)$. For example, if you have some ciphertext $c = (g^r, my^r)$, you can re-randomize by creating $c' = (g^rg^s, my^ry^s) = (g^{r+s}, my^{r+s})$. The new coefficient that you randomly chose is now $r + s$.

Unfortunately, malleability is not the best for some cases. If we want to create something which is IND-CCA2 secure, we need to exclude malleability. The next scheme (Cramer-Shoup) builds off of ElGamal and creates a scheme which is IND-CCA2 secure.

3.1.1. *Cramer Shoup.* Let G_q be a group of prime order q. For example, we could use $G_q = Q_p$ where $p = 2q + 1$ and p, q are prime.

Key Generation:

- Pick two number $g_1, g_2 \leftarrow G_q$ uniformly at random from the group.
- Pick five numbers $x_1, x_2, y_1, y_2, z \leftarrow Z_q$ uniformly at random from the integer group.
- Set $c = g_1^{x_1}g_2^{x_2}$, $d = g_1^{y_1}g_2^{y_2}$, and $h = g_1^z$.
- Output $PK = (g_1, g_2, c, d, h)$ and we obtain $SK = (x_1, x_2, y_1, y_2, z)$.

Encryption:

- Select some number $r \leftarrow Z_q$ uniformly at random.
- Set $u_1 = g_1^r$ and $u_2 = g_2^r$.
- Compute $e = h^rm$ and $\alpha = H(u_1, u_2, e)$ where $H : G_q^e \to Z_q$.
- Set $v = c^rd^{r\alpha}$ and return the ciphertext (u_1, u_2, e, v).

Decryption:

- Using H, compute $\alpha = H(u_1, u_2, e)$.
- Check that $u_1^{x_1+y_1\alpha}u_2^{x_2+y_2\alpha} = v$. If not equal, reject the message. If equal, return $m = eu_1^{-z}$ as the message.

Proof of Correctness. Note that $eu_1^{-z} = h^rmu_1^{-z} = h^rm(g_1^r)^{-z}$. But we have chosen h such that $h = g_1^z$ so that $h^rm(g_1^r)^{-z} = h^rm(h^{-1})^r = m$. Now to give intuition on the correctness condition, we note that:

$$(u_1^{x_1}u_2^{x_2})(u_1^{y_1\alpha}u_2^{y_2\alpha}) = ((g_1^r)^{x_1}(g_2^r)^{x_2})((g_1^r)^{y_1\alpha}(g_2^r)^{y_2\alpha}) \tag{1}$$
$$= c^r(d^r)^\alpha \tag{2}$$

But we have defined v such that $v = c^rd^{r\alpha}$, so we are merely checking that this condition is true.

3.2. **RSA.** RSA is a public key encryption system.

Key Generation: We define $Keygen(1^\lambda)$ as the following process:

- Pick two large primes p, q and set $n = pq$.
- Pick some number $e \leftarrow Z_{\phi(n)}^*$ and compute $d = e^{-1} \pmod{\phi(n)}$. Note that picking e is equivalent to picking some e from Z_n^* where $gcd(e, \phi(n)) = 1$.
- Set the keys as $PK = (n, e)$ and $SK = (d, p, q)$.

Encryption: We have some message $m \in Z_n$ and we define the encryption operation as $Enc(PK, m) = m^e \pmod{n}$.

Decryption: We have some ciphertext c and define decryption as $Dec(SK, c) = c^d \pmod{n}$.

Proof of correctness: Note that by CRT, proving correctness for $(m^e)^d \pmod{n}$ is equivalent to proving correctness for $(m^e)^d \pmod{p}$ and $(m^e)^d \pmod{q}$. WLOG we will prove it correct for p. In this case, if $m = 0 \pmod{p}$, then $(m^e)^d \equiv 0 \pmod{p}$ so we have the relation $m = (m^e)^d \pmod{p}$. If $m \neq 0 \pmod{p}$, then we have $(m^e)^d = m^{ed} \pmod{p}$. But we know that $d = e^{-1} \pmod{\phi(n)}$ so that $ed = 1 \pmod{(p-1)(q-1)}$. This implies that $ed = 1 + t(q-1)(p-1) = 1 + u(p-1)$. Therefore, we have $m^{ed} = m^{1+u(p-1)} \pmod{p}$ and by FLT, we have $m^{ed} = m \pmod{p}$. Implies that $(m^e)^d = m \pmod{p}$, which completes the proof.

The basic assumptions underlying RSA is that the RSA problem is hard, i.e that given c and e the computation of $c = m^e \pmod{n}$ is difficult for n large. The best known current way to solve this problem is by factoring n and solving the congruence modulo smaller primes, via the Chinese Remainder Theorem.

3.2.1. *RSA-OAEP.* Regular RSA is not semantically secure (i.e IND-CPA) and its not even randomized. Therefore, it's not IND-CCA2 secure either. In order to get it to IND-CCA2, people have developed a new method for encryption which builds off of RSA. The new method adds some extra bits of padding (in order to check validity) and also some extra random bits. The algorithm is a feistel network which uses random oracles G and H to create a new plaintext message to send into RSA.

Functions have the mapping $G : \{0,1\}^{k_0} \to \{0,1\}^{t+k_1}$ and $H : \{0,1\}^{t+k_1} \to \{0,1\}^{k_0}$, where $t = |m|$ is the length of the message.

Encoding using RSA-OAEP:

- Start with k_0 number of random bits to add and k_1 number of extra bits to add to end of message. We have two starting messages $X_0 = m||0^{k_1}$ and $Y_0 \leftarrow \{0,1\}^{k_0}$.
- Obtain the result $X = X_0 \oplus G(Y_0)$.
- Use the result X to compute $Y = H(X) \oplus Y_0$.
- The message $m' = X||Y$ is now passed into RSA and encrypted to become $(m')^e \pmod{n}$.

Notice that this encoding makes sure to include some random bits Y and does some transformations in the middle so that the bits are scattered about. Decoding is easy:

- Take the ciphertext $c' = (m')^e \pmod{n}$ and decrypt regularly in RSA (i.e. $m' = c'^d \pmod{n}$).
- Parse the decrypted plaintext for X and Y.
- Obtain Y_0 by doing $Y_0 = H(X) \oplus Y$.
- Use the result of Y_0 to compute $X_0 = G(Y_0) \oplus X$.
- Check to make sure that the last k_1 bits in X are all zeros. If this is not true, discard the message as invalid. If it is true, accept the message.

RSA-OAEP is IND-CCA2 secure, assuming ROM for G and H and RSA is hard to invert on random inputs.

4. Diffie-Hellman Key Exchange