6.857
NETWORK AND COMPUTER SECURITY
QUIZ REVIEW AND NOTES

LECTURER: RONALD RIVEST
SCRIBE: JOHN WANG

Contents

1. One Time Pad

1.1. **Basic Secret Key Encryption.** In the most basic form of encryption, you have a secret key K and you have the following operations in your cryptosystem:

- $Keygen(1^\lambda) \to K$ generates a secret key K.
- $Enc(K, m)$ encrypts a message m into a ciphertext c.
- $Dec(K, c)$ decrypts a ciphertext c into the original message m.

1.2. **One Time Pad.** The one time pad (OTP) is the simplest form of encryption and decryption. You have a secret key K and message m which are both the same length so that $|K| = |m| = \lambda$. You generate the secret key K by randomly generating λ bits so that $K \leftarrow \{0,1\}^\lambda$.

Once we have created the secret key K, we can encrypt a message m by simply setting the ciphertext as $m \oplus K = c$. Decryption is easy because XOR is an invertible operation so that $m = K \oplus c$. The operations are:

- $Enc(K, m) = K \oplus m$.
- $Dec(K, c) = K \oplus c$.

1.3. **Proof of Security of OTP.** *Theorem:* OTP is unconditionally secure.

Definition: Unconditional security is security regardless of the computing power and resources that an adversary might have, i.e. that a scheme is information-theoretically secure. The scheme should be secure even when the adversary has unlimited computing power.

Proof of Unconditionally Security: We want to show that $P(M) = P(M|c)$ so that Eve's prior probability that the message is M is the same as Eve's posterior probability that the message is M given that Eve sees ciphertext c.

We compute $P(M|c) = P(c|M)P(M)/P(c)$. We want to compute $P(c|M)$, which is the probability that we have ciphertext c given that the message is M. This is just the probability that the key is equal to $k = c \oplus M$. Since each key k is equally likely, we have $P(c|M) = P(k) = 2^{-\lambda}$. To compute $P(c)$, we know that $P(c) = \sum_M P(c|M)P(M) = \sum_M 2^{-\lambda} P(M) = 2^{-\lambda} \sum_M P(M) = 2^{-\lambda}$. Therefore, we see that $P(M|c) = 2^{-\lambda} P(M)/2^{-\lambda}$.

Thus we see that $P(M) = P(M|c)$, which completes the proof.

2. Hash Functions

2.1. **Random Oracle Model (ROM).** The ROM is an idealized version of a hash function. It's been proven to be unachieveable in practice. The Oracle receives input $x \in \{0,1\}^*$ and outputs $h(x)$ such that $|h(x)| = d$ bits for some fixed d. The construction is as follows:

- For a given input $x \in \{0,1\}^*$, if x is not in the book then the oracle flips a random coin d times and returns the sequence of heads. Oracle records $(x, h(x))$ in the book.
- If x is in the book, the oracle returns $h(x)$.

Thus, we see that $h(\cdot)$ is deterministic but returns a random answer for new inputs.

2.2. **Properties.** There are a number of properties that a good hash function should have (each property is dependent on what the function is being employed to do):

- (OW) One-wayness. Preimage resistance.
- (CR) Collision resistance. Strong collision resistance.
- (TCR) Target collision resistance. Weak collision resistance. Second preimage resistance.
- (NM) Non-malleability.
- (PRF) Pseudo-randomness.

Theorem: If hash function is collision resistant, then it is target collision resistant as well. The proof is by contradiction. If not target collision resistant, then you automatically find a collision. Converse is not true.

Theorem: If h is one-way, then one does not necessarily know if h is collision resistant. The converse is also true. For example, consider $h(x) = x$ which is collision resistant but not one-way.

2.3. **Merkle Tree.** To authenticate a collection of n objects, you can do better than just authenticating everything. This is important in file systems, time-stamping data, etc.

You create a merkle tree where the parent of values x_i, x_{i+1} is $h(x_i||x_{i+1})$. You perform this recursively to create a tree of height $\log n$.

To authenticate a value x_i, you give the sibling of x_i and move up the tree, recursively authenticating the sibling of x_i all the way up to the root. Authentication passes if authentication works for each value and its sibling. You need collision resistance for this to work, otherwise someone could create a new $x' = h(x'_1||x'_2) = h(x_1, h_2)$.

2.4. **Merkle-Damgard Construction.** We create a mode of operation for dealing with fixed length input on a hash function f, if the f takes only a specified input. We specify the following parameters:

- Output size d.
- Chaining variable size c.
- Message block size b.
- Compression function $f : \{0,1\}^b \times \{0,1\}^c \to \{0,1\}^c$.
- Initialization Vector IV which is c bits in length.

From the above parameters, we can create a chain of functions which can handle an input of any length. We pad the input m with 10^* and a fixed-length representation of the length of the message so we have a new message $M = m||10*|||m|$. From there, we break M into smaller blocks of size b so we have $M_1, M_2, \ldots$.

To hash, we start with $IV = c_0$ and we provide the function f with inputs c_0 and M_1. This produces c_1. The general formula is:

$$f(c_{i-1}, m_i) = c_i \tag{1}$$

We take the last result c_n as the hash that we output for message m so that $h(m) = c_n$.

Note that if f is collision resistant or one-way, then so is h. Suppose not, then there exists some collision for h, and we can work our way back until we get to the first call to f, which leads to a contradiction. Same reasoning for one-wayness.

3. 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.

3.1. 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.

3.2. 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.

3.3. 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.

3.4. 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

4. Block Ciphers

We use block ciphers when we have a block of plaintext and want to encrypt it into a block of ciphertext. The idea behind this is that if you have a very large file, then you don't want to encrypt the entire thing, but rather parts of it. There are many "modes of operation" of block ciphers which handle variable length input.

The basic way that a block cipher operates is by having some plaintext block p and encrypting it with some key k to obtain ciphertext c. The modes of operation provide an information service, such as confidentiality or authenticity.

4.1. **Practical Block Ciphers.**

4.1.1. *Data Encryption Standard (DES).* This is an outdated mode of operation, but was one of the original modes of operation. Works on inputs of 64 bit plaintexts using 16 rounds. It uses a feistel structure to shuffle the bits around:

- Divide the input into 32 bit halves (l_0, r_0).
- Set $l_1 = r_0$ and compute $r_1 = l_0 \oplus f(r_0, k_0)$, where f encrypts r_0 using key k_0.
- Continue this for 16 rounds, each time using a new round key k_i.

The final result l_{16}, r_{16} is concatenated back together to give the ciphertext. Inversion is easy because you can first take l_{16} and compute $f(r_{16}, k_{16}) \oplus r_{16}$ to get l_{15}. Continue backwards until you get l_0, r_0.

This mode is broken because it is subject to differential and linear attacks, and because the key is too short.

4.1.2. *Advanced Encryption Standard (AES).* The standard that replaced DES. Uses 128-bit plaintext/ciphertext blocks and keys of size $128, 192$, or 256 bits in length. Can have $10, 12$, or 14 rounds depending on key length.

Basically, this encryption standard uses a byte oriented design and views the 128 bit input as a 4 by 4 array of 16 bytes. The array is permutated with four different operations, and these permutations performed during each round. The permutations are as follows:

- XOR the current state with the round key.
- Substitute bytes from a lookup table (non-linear operation).
- Rotate rows (each by different amounts).
- Mix each column (by a linear operation).

Output the final state after however many rounds you are running. Can think of AES as an ideal block cipher, since it is pretty secure.

4.2. **Common Modes of Operation.**

Modes of operation use an ideal block cipher (AES for example) and construct a system for encrypted inputs of variable length which are possibly longer than a single block.

4.2.1. *ECB (Electronic Code Book).* Simplest mode of operation. Divide the input into blocks of b bits in length, and encrypt each block with the same key. If the input is not the right length, you can pad the input with a single 1 bit, and enough 0s to make length a multiple of b.

ECB preserves a lot of patterns and is really only good for sending random data.

4.2.2. *CTR (Counter Mode).* Generate a pseudorandom sequence by starting at some integer i and using the sequence $i, i+1, \ldots$. For each integer in this sequence, we encrypt the integer using a block cipher to get $x_i, x_{i+1}, \ldots$. The resulting sequence of x_i's is a pseudorandom sequence, and we can obtain ciphertexts using $c_i = m_i \oplus x_i$.

We can then transmit $i, c_1, c_2, \ldots$, and the receiver can decrypt these by just recreating the sequence of $x_i, x_{i+1}, \ldots$ and XORing with the received ciphertexts $c_1, c_2, \ldots$.

4.2.3. *CBC (Cipher Block Chaining).* We choose an initial value IV which we shall use to create a new message for encryption in the first round. Take $m_1 \oplus IV$ and encrypt it with key K to obtain c_1. Now encrypt m_2 using $m_2 \oplus c_1$ to obtain c_2. They key remains the same throughout all of the rounds.

General form of this encryption is as follows: $c_i = Enc(K, c_{i-1} \oplus m_i)$ for all $i > 1$ and $c_1 = Enc(K, IV \oplus m_1)$ for $i = 1$. The ciphertext transmitted is $IV, c_1, c_2, \ldots$.

Decryption is fast and parallelizable because you only need to know previous c_{i-1} to decrypt c_i. To decrypt, you just do the inverse decryption on c_i and XOR the result with c_{i-1} to get m_i. As a formula, this is $Dec(K, c_i) \oplus c_{i-1} = m_i$.

The last block c_n is the CBC-MAC message for message m.

4.2.4. *CFB (Cipher Feedback).* This mode chooses an initial value IV and encrypts this value using key k. The result is XORed with the first message block to obtain $c_1 = m_1 \oplus Enc(k, IV)$. The next ciphertext block is created by taking $m_2 \oplus Enc(k, c_1)$, etc.

The general formula for the CFB is then $c_i = m_i \oplus Enc(k, c_{i-1})$ for $i > 1$ and $c_1 = m_1 \oplus Enc(k, IV)$ for $i = 1$.

4.2.5. *UFE (Unbalanced Feistel Encryption).* None of the previous block-cipher modes were IND-CCA secure. This is because you can just choose $m_0 = 0^\lambda$ and $m_1 = 1^\lambda$. Then, encrypt $1^{\lambda-1}$ and use a length extension attack. All of the previous modes would provide you with the first part of the ciphertext, and so would fail IND-CCA security. The UFE mode attempts to prevent this.

Setup: We have a key $K = (k_1, k_2, k_3)$ and a random value $r \leftarrow \{0, 1\}^\lambda$.

Encryption: The random value r is the starting value for CTR mode, and we use key k_1 to encrypt each of these values to obtain a sequence of pads $p_1, p_2, \ldots, p_n$ where $p_i = Enc(k_1, r + i)$. Next, we obtain the cipertext by finishing CTR mode, i.e. using $c_i = m_i \oplus p_i$. However, the MAC is different than in CTR mode. Instead, we use the c_i's and send it through CBC-MAC with keys k_2, k_3 to obtain a sequence x_i. The IV for the CBC-MAC is just 0^λ so we have $x_i = Enc(k_2, x_{i-1} \oplus c_i)$, $x_n = Enc(k_3, x_{n-1})$, and $x_1 = Enc(k_2, 0^\lambda)$. We calculate $\sigma = r \oplus x_n$ and output $\sigma, c_1, c_2, \ldots, c_n$.

Decryption: Decryption of the scheme involves first computing the sequence of x_i's again using c_i. Once this sequence is computing, you can find r by using $r = x_n \oplus \sigma$. Finally, send r through CTR mode to recompute the pad p_i, and use $m_i = p_i \oplus c_i$ to recover the message.

The resulting UFE mode is IND-CCA secure, but it is only designed for confidentiality. There is no way to tell if the ciphertext has been changed at all, for that you need a message-authentication protocol.

5. Message Authentication Codes (MACs)

Often, you want to be able to provide integrity (not just confidentiality). This means that you want to be able to send a message and have the receiver of that message be sure that the message wasn't tampered with on its way over.

This is an orthogonal concept to confidentiality, and you typically do both by appending a MAC to the ciphertext message. The game that Alice, Bob, and Eve play is as follows:

- Alice computes $Mac(k, m)$ and appends it to m.
- Bob recomputes $Mac(k, m)$ and verifies it is correct, rejects if the computed is different than the received MAC.
- Adversary wants to forge $m', Mac(k, m')$ and may hear valid MACs previously sent (but she wants to forge to a new message m' she has never seen).

5.1. **MAC using Random Oracle (PRF).** If you use a pseudorandom function family (PRF) to create a MAC, then all you have to do is use $Mac(k, m) = h(k||m)$. This holds only if h is indistinguishable from a random oracle. Note that for sequential hash functions, this will not work and you need to take special care.

5.2. **HMAC.** One common method for creating an authentication code is to just use a cryptographic hash function in the HMAC protocol. Take $k_1 = k||opad$ and $k_2 = k||ipad$ where *opad* and *ipad* are fixed constants. Then you can use:

$$HMAC(k, m) = h(k_1||h(k_2||m)) \tag{2}$$

This is good because it scrambles m and prevents length extension attacks.

5.3. **CBC-MAC.** Use the last block of the CBC block cipher mode. However, instead of using the original key k_c to encrypt the last block mode, you must use some new key $k' = k \oplus c$ where c is a constant and k is the key for the CBC-MAC. Then, you take the last encrypted block in CBC-MAC and encrypt it again with k' in order to get CBC-MAC.

6. Digital Signatures

The idea of the digital signature is to switch the roles of the secret key and the public key. We create a signature for the message that anyone can decrypt, but only one person can encrypt. Thus, encrypt the signature with the secret key and decrypt with the public key.

6.1. **Description of Digital Signatures.** Creating a digital signatures a standard set of procedures:

- $Keygen(1^\lambda) \to (PK, SK)$.
- $Sign(SK, m) \to \sigma_{SK}(m)$ creates a signature (which may be randomized).
- $Verify(PK, m, \sigma)$ results in a boolean value.

A digital signature is correct if for all m, we have $Verify(PK, m, Sign(SK, m)) = True$.

6.2. **Security of Digital Signature Schemes.** Just like other security definitions, the security definition for digital signature schemes is created in terms of a game. We say a digital signature scheme has *(weak) unforgeability under adapative chosen message attack* if adversary has no advantage in the following game:

- Examiner obtains (PK, SK) from $Keygen(1^\lambda)$ and sends PK to the adversary.
- Adversary obtains signatures to a sequence of messages $m_1, m_2, \ldots, m_q$ of his choice where $q = poly(\lambda)$. We allow the adversary to choose m_i based on information obtained in messages $m_1, \ldots, m_{i-1}$.
- Adversary outputs a message/signature pair (m, σ_*).

The adversary wins this game if $Verify(PK, m, \sigma_*) = True$ and $m \notin \{m_1, \ldots, m_q\}$. Scheme is secure if the probability of the adversary winning is negligible.

The digital signature scheme is *strongly secure* if the adversary can't produce a new signature (m, σ_*) for a message m that he has previously seen (i.e. $m \in \{m_1, \ldots, m_q\}$).

6.3. **Message Compression with Hash Functions.** Often you don't want to sign the entire message m, but would rather sign a smaller number. You could use a cryptographic hash function to create $h(m)$ which is the desired length, and change $Sign(SK, m) = Sign(SK, h(m))$. In this, you need collision resistance to prevent someone from getting a message m' and fake that you have signed m' when you really only signed m. Notice that you don't need one-wayness because it's not necessary for you to hide m given $h(m)$.

7. Commitment Schemes

Commitment schemes are schemes in which people give a bid and commit to that bid beforehand, yet other people are unable to see that bid until the committer of the bid reveals it. This is especially useful in auctions. You obviously would want some particular properties. These properties are:

- Hiding. $Commit(x)$ should not be visible to anyone and one should not be able to gain information about x from $Commit(x)$.
- Non-malleability. Can't produce a related commitment from $Commit(x)$. For example, you shouldn't be able to get $Commit(x+1)$ from $Commit(x)$.
- Binding. You can only open $Commit(x)$ and reveal the original x which you committed with.

In commitment schemes, we have two functions $Commit(x)$ which creates a commitment to x and $Reveal(c)$ which reveals the commitment c.

7.1. Hash Function Commitment Scheme.

The simplest way to produce a commitment scheme is to have a good cryptographic hash function. Let's say we have some value x that we want to seal. The sealer chooses some random value $r \leftarrow \{0,1\}^b$ where b is some sufficiently large number and computes $Commit(x) = h(x||r)$.

For this scheme to work, we need the following hash function properties:

- One wayness. If you could turn invert $Commit(x)$ into $h(x||r)$, then you would be able to obtain x, and you wouldn't have the hiding property.
- Target collision resistance. Given $h(x||r)$, you don't want someone to be able to produce a collision with $x||r$ using some $x'||r'$, otherwise the non-malleability property would be lost.
- Non-malleability.

7.2. Pedersen Commitment Scheme.

Setup: Choose p, q as large primes such that $q|p-1$ and p is a safe prime. For example $p = 2q+1$ would work. Now let q be the generator of the order q subgroup of Z_p^*. Let $h = g^a$, where a is secret.

Commit: We want some commitment x where $x \in Z_q$. Then we choose $r \in Z_q$ uniformly at random and set $Commit(x) = c = g^x h^r \pmod{p}$.

Reveal: The sender reveals x and r, and the receiver verifies that $c = g^x h^r \pmod{p}$.

To make sure all the properties are satisfied, let's go through them:

Hiding: Given $c = g^x h^r \pmod{p}$, this doesn't tell us anything about x because any x' could be possible. Just pick some $x' \in Z_q$ and we can find an r' that matches this x' in the following way:

$$g^x h^r = g^{x'} h^{r'} \pmod{p} \tag{3}$$
$$g^x g^{ar} = g^{x'} g^{ar'} \pmod{p} \tag{4}$$
$$g^{x+ar} = g^{x'+ar'} \pmod{p} \tag{5}$$
$$x + ar = x' + ar' \pmod{q} \tag{6}$$
$$r' = (x - x')/a + r \pmod{q} \tag{7}$$

Binding: Let's see if the sender can reveal the message c in two different ways. He can't, as long as the DLP is hard, because he needs to compute a in order to get r' in the above equation. For a, you must solve $h = g^a \pmod{p}$.

Non-malleable: Pedersen doesn't satisfy this. For instance, consider $c = g^x h^r$ and $c' = gc$, then we have $c' = g^{x+1} h^r$ so that $Commit(x+1) = Commit(x)g$. This is clearly mealleable.

8. Cryptographic Systems

8.1. **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 $[0, \dots, |G|-1]$ and set $SK = x$ and $PK = g^x$.

Encryption: Pick $k \leftarrow [0, \dots, |G|-1]$ 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 $c_1 = (g^r, m_1 y^r)$ and $c_2 = (g^s, m_2 y^s)$, then we have $c_1 c_2 = (g^{r+s}, (m_1 m_2) y^{r+s}) = Enc(m_1, m_2)$.

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 $c' = (g^r g^s, my^r y^s) = (g^{r+s}, my^{r+s})$. 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.

8.1.1. *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 $e = h^r m$ and $\alpha = H(u_1, u_2, e)$ where $H : G_q^e \to Z_q$.
- Set $v = c^r d^{r\alpha}$ 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 $eu_1^{-z} = h^r m u_1^{-z} = h^r m (g_1^r)^{-z}$. But we have chosen h such that $h = g_1^z$ so that $h^r m (g_1^r)^{-z} = h^r m (h^{-1})^r = m$. Now to give intuition on the correctness condition, we note that:

$$
\begin{aligned}
(u_1^{x_1} u_2^{x_2})(u_1^{y_1 \alpha} u_2^{y_2 \alpha}) &= ((g_1^r)^{x_1} (g_2^r)^{x_2})((g_1^r)^{y_1 \alpha} (g_2^r)^{y_2 \alpha}) && (8) \\
&= c^r (d^r)^\alpha && (9)
\end{aligned}
$$

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

Notes: Cramer-Shoup satisfies IND-CCA2 security if DDH and if H is target collision resistant. Our strongest notion of security is achievable, but at some cost in terms of speed and conplexity of the algorithm.

8.2. **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.

8.2.1. *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.

9. Diffie-Hellman Key Exchange

9.1. **Key Exchange Algorithm.** The key idea behind Diffie-Hellman key exchange is that you want to establish a secret key between Alice and Bob, without Eve being able to listen in. If Eve can hear all communication that happens between Alice and Bob, this is somewhat hard, and you need to have a particular scheme that will prevent Eve from obtaining too much information. The Diffie-Hellman algorithm relies on the difficulty of the CDH problem in finite abelian groups.

Setup: We have a cyclic group $G =< g >$ with generator g. Both G and g are fixed and public. Alice picks some secret $x \leftarrow [0, \dots, |G| - 1]$ are random, and Bob picks some secret $y \leftarrow [0, \dots, |G| - 1]$ at random.

Exchange: Alice and Bob set their public keys as g^x, g^y respectively and communicate these to each other.

Computation of Shared Secret: To compute the shared secret key, Alice takes Bob's secret key of g^y and exponentiates to get $K = (g^y)^x = g^{xy}$. Likewise, Bob takes Alice's secret key g^x and exponentiates to get $K = (g^x)^y = g^{xy}$ which is a shared secret.

If the adversary doesn't know either x or y, then it is hard for him to get g^{xy} based on the difficulty of CDH.

9.2. **Computational Diffie-Hellman (CDH).** This assumption arises from the Diffie-Hellman key exchange. We assume that if we are given g^a, g^b, it is computationaly infeasible to compute g^{ab}.

9.3. **Decisional Diffie-Hellman (DDH).** There is a very related problem to CDH, called DDH. The DDH problem states that given a group G with generator g, it is hard/infeasible to decide whether a given triple of elements was generated:

$$(g^a, g^b, g^c) \text{ vs. } (g^a, g^b, g^{ab}) \tag{10}$$

Where a, b, c are all randomly chosen from the group G.

If DDH holds in a group, then you can't even recognize when the group key g^{ab} is given to you.

9.4. **Connection Between DDH and CDH.**

9.4.1. *DDH implies CDH. Theorem:* If DDH holds, then CDH holds as well.

Proof: Given that DDH holds, suppose that CDH does not hold. Then given g^a, g^b, it is not computationally intractable to compute g^{ab}, and so we can compute g^{ab} in polynomial time. Then, this means that we can distinguish between g^c and g^{ab} if $c \neq ab$ because we can just check if $g^k = g^{ab}$ for some g^k. This is in contradiction to DDH. Therefore, we see that if DDH holds, then CDH must hold as well.

9.4.2. *CDH is weaker than DDH.* It is believed that CDH is a weaker assumption than DDH. That is, there are groups for which detecting DDH tuples is easy, but for which solving CDH problems is hard.