Cryptography A - Lecture Notes

Shared

June 5, 2013

CONTENTS

Chapter 1

INTRODUCTION

1.1 Cryptography and Modern Cryptography

Classical cryptography was, until the 20th century, considered more of an art than a science as it had no real theory to rely upon.

Modern cryptography, however, is much more formal and encompasses a great deal more than just secret communication. It also covers: message authentication, digital signatures, electronic auctions and elections, digital currency etc.

It also differs by the places in which it is found; Classical cryptography was mainly used by the military and intelligence organisations, whilst modern cryptography is used in almost every computer system. It is used when accessing a secure website, when authenticating on a multi-user operating system etc.

Definition: **Modern Cryptography is the scientific study of techniques for securing digital information, transactions, and distributed computations.**

1.2 Principles of Secure Communication

There are two types of encryption systems: symmetric (**private key**, or classical) and asymmetric (**public key**) encryption. Both rely on these principles for secure communication.

Confidentiality: An adversary cannot see which messages are transmitted, though they may see that something is being sent.

Authenticity: An adversary cannot cause non-original messages to be accepted. This is related to but different from...

Integrity: An adversary cannot alter messages in transit without the alteration being detected.

1.3 Private Key Encryption

Private-key encryption implies both the sender and the receiver use the same secret key to encrypt and respectively decrypt the message.

This method was useful for the military in the past as the two parties would be able to physically meet and decide upon a key whereas today achieving that is more difficult as the parties meeting would be inconvenient or impossible most times (think about online money transfers or user authentication).

However, this method is still being applied in some scenarios like disk/file encryption where the same user at different points in time uses the same secret key to both read and write to a file. Private-key encryption is also widely used in conjunction with asymmetric methods.

The private key encryption scheme is comprised of three algorithms:

- The key generation algorithm(Gen) which is a probabilistic algorithm that outputs a key, k, chosen according to some distribution that is determined by the scheme.
- The encryption algorithm(Enc) which takes as input a key k and a plaintext message M and outputs a cipher text C: $Enc_k(M) = c$;
- The decryption algorithm(Dec) which takes as input a key k and a cipher C and outputs the original message M: $Dec_k(C) = m$;

Decrypting a cipher text using the appropriate key yields the original message or: $Dec_k(Enc_k(m)) = m$.

1.4 Kerckhoff's Principle

Kerckhoff's principle is fundamental to cryptography. It is this: **"Security should rely solely on the secrecy of the key."**.

Arguments:

- It is much easier to maintain secrecy of a short key than a complex algorithm.
- Changing a disclosed key is much easier than changing a disclosed algorithm.
- In case more parties need to communicate it is easier to produce multiple keys than multiple algorithms.

Example of attack scenarios:

- Cipher-only attack
- Known-plaintext attack
- Chosen-plaintext attack
- Chosen-cipher attack

Chapter 2

CLASSICAL CRYPTOGRAPHY

2.1 Historical Ciphers and Their Cryptanalysis

- **Caesar cipher**

  The encryption is done by rotating the letters of the alphabet by 3(A becomes D). The biggest problem with this cipher is that the method is fixed and there is no key to speak of. Therefore anyone learning the decryption algorithm would be able to get the initial message effortlessly.

- **The shift cipher**

  This cipher is similar to the Caesar cipher but it introduces a secret key k that is a number between 0 and 25. The encryption is then done by shifting the letters by k places.

  $Enc(m_i) = [(m_i + k) \mod 26];$
  $Dec(c_i) = [(c_i - k) \mod 26]$

  This cipher is not secure as it can be easily broken by doing an exhaustive search. There are only 26 possible keys therefore the cipher does not follow the insufficient key space principle which says that any secure encryption scheme must have a key space that is not vulnerable to exhaustive search. The number of possible keys must be very large (at least 2^{60} or 2^{70}).

- **Mono-alphabetic substitution**

  The idea behind the mono-alphabetic substitution is to map each plaintext character to a different cipher text character in an arbitrary manner. The keys space thus consists of all permutations of the letters of the alphabet which is approximately 2^{88}, too big for an exhaustive search.
  However this does not make it secure. Considering that the plain text message is written in plain English we can calculate the frequency of each letter and then compare the cipher‘s table of frequencies with the table of frequencies for the English language.

- **Improved attack on the shift cipher**

The initial approach was to try all possible keys and check which one gives a message that "makes sense". But sometimes it is hard to define what "makes sense" actually means. If the initial message is not written in plain English it is hard to know which solution is the correct one. There are cases though when the plain text message is not written in valid English but has the letter distribution of such a text. In this case we compute the following sum: $I_j = \sum_{i=0}^{25} p^2 * \approx 0.065$ where p_i is each letter's frequency in the plain text with $0 <= i <= 25$. We know that the letters are shifted by k spaces, therefore $q_{i+k} = p_i$. Now all we need to do is compute the sum $I_j = \sum_{i=0}^{25} p_i * q_{i+j}$ where q_i is each letter's frequency in the cipher text and compare it to 0.0065. The j for which $\mathrm{I_j}$ is closest to 0.065 is the key.

- **The Vigenere cipher**

  The statistical attack on the mono-alphabetic substitution cipher was possible because the mapping of each letter was fixed.
  Consider the possibility when a letter is mapped to multiple letters. In this case the table of frequencies would be irrelevant as the frequencies are very similar. The Viginere cipher takes a repeated small string and forms the key which is then added to the original message.
  In order to simplify the breaking process we first need to identify the length of the key. If the original message is written in plain English then we should look for repeated sequences of length 2 and 3 in the key. These will indicate us the appearances of common words like *"the"* which are in the same relative position of the small substring that is repeated, therefore the distance between 2 such appearances would be a multiple of the length of the substring. Knowing this we can assume that the greatest common divisor between the distances of such appearances is either the length of the key or a multiple of it.
  Knowing the key's length the task is much simplified. We notice that we can split the initial text in a number of substrings equal to the keys length(n). This way the characters on positions *1,n, 2n* etc. will be encrypted using the first character of the key, the positions *2, n+1, 2n+1* etc. using the second character of the key and so on.
  We now have n different ciphers that are encrypted using the shift cipher with key *K[i]*. Therefore we need to decipher each string individually. Considering that we selected dispersed characters from a plain English text, all the substrings will not be in plain English. Knowing this we can now apply the method described above under *"Improved attack on the shift cipher"*.

2.2 Perfectly Secret Encryption

Definition: An encryption scheme (Gen, Enc, Dec) over a message space m is perfectly secret if for every probability distribution over m, every message $m \in M$, and every ciphertext $c \in C$ for which $P(C = c) > 0$:

$$P(M = m|C = c) = P(M = m)$$

In other words a scheme is perfectly secret if the distributions over messages and ciphertexts are independent.

2.3 One Time Pad

The one time pad is a good system to review because it is **totally secure** / perfectly secret, if used correctly. Sadly, it is difficult to use in the real-world.

It works by XOR-ing the entire message by the cipher. This means that the cipher must be as long as the message, which is clearly a limitation in the real world — encrypting your hard drive would require another hard drive of data for the cipher.[1]

The other limitation is the fact it can only be used once. This is because an adversary can ask you to encrypt a message ($c = m \oplus k$). They can then use that to calculate the key, k, because they have both c and m ($k = m \oplus c$). In practice, this means that if you know part of a message being sent, you can use that to calculate the bits of cipher in those positions.

2.4 Shannon's Theorem

Shannon's theorem provides a generalised definition for deeming whether a cryptosystem is **perfectly secret**. First, we assume we have a system with equal numbers of keys, plaintext, and ciphertexts; $\#\mathbf{K} = \#\mathbf{P} = \#\mathbf{C}$.

We say that this system provides perfect secrecy iff:

- The probability of any key being used is $1/\#K$.
- For each $m \in P$ there exists a $c \in C$.

It's worth briefly looking into the *why* of Shannon's theorem. If $\#K = \#P = \#C$, then given a key, there must be one mapping of each plaintext to each ciphertext. And if we try every key, then a given message must map to every ciphertext. If it does not, then we have leaked information about the message, for example if m_1 maps to c_{666} more than once when encrypted with every key, then given c_{666} we can say of all messages, it is more likely to be m_1 than another message, despite not even knowing the key!

[1]The XOR operations outputs a 0 bit at every position in which both operands have the same bit, and a 1 in positions where they differ.

Chapter 3

MODERN CRYPTOGRAPHIC DEFINITIONS

3.1 The Three Principles of Modern Cryptology

1. Formulate a rigorous and **precise definition of security**.
2. When a cryptographic system relies on an **unproven assumption**, it must be **precisely stated**. Furthermore, this assumption **must be minimal**.
3. Cryptographic constructions must be accompanied by a **rigorous proof of security** with respect to the definition formulated in principle 1, revolving around the assumptions stated in principle 2.

3.1.1 Principle 1: Formulation of Exact Definitions

- **Why is this important?**
  - **Importance for design:** We need to have a good understanding of our goal so that we know when we achieve it. Sometimes our cryptographic system doesn't need to be as efficient as possible, a simpler version being sufficient.
  - **Importance for usage:** When we want to choose an existing cryptographic scheme we need to have some sort of comparison criteria to be able to tell if it suffices for our application.
  - **Importance for study:** For researching different types of cryptographic systems we need to be able to compare them and measure their security level. Without a rigorous proof of security the only performance attribute we can measure is efficiency (which is not a very accurate measure of security).
- **How do we define security?**
  - An encryption scheme is secure if no adversary can compute any function of the plaintext from the cipher text.
  - An encryption scheme is considered broken if an adversary learns some function of the plaintext from the cipher text.
  - The power of the adversary relates to assumptions regarding the actions the adversary is assumed to be able to take, as well as the adversary's computational power.

3.1.2 Principle 2: Reliance on Precise Assumptions

Most cryptographic constructions cannot be proven secure unconditionally, therefore it relies on some assumptions which must be precisely stated for the following reasons:

1. **Validation of the assumption** If the assumption being relied upon is not precisely stated and presented it cannot be studied and potentially refuted.
2. **Comparison of schemes** If the assumptions used by two schemes are incomparable, then the one based on the better-studied or the simpler assumptions is preferred.
3. **Facilitation of proofs of security** A mathematical proof that "the construction is secure if the assumption holds" cannot be provided without a precise statement of what the assumption is.

3.1.3 Principle 3: Rigorous Proofs of Security

The first two principles lead naturally to this one. Most proofs in modern cryptography use the **reductionist** approach, that is, *"Given the Assumption A is true, Construction C is secure according to the given definition."*

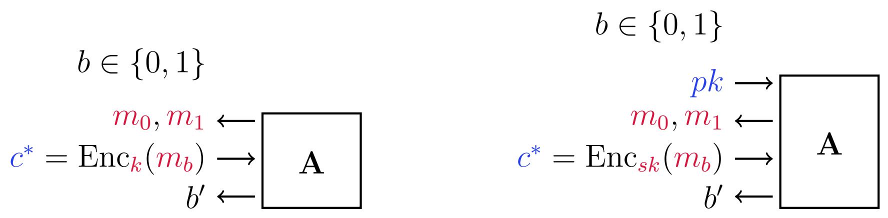
3.2 Cryptographic Games

We can capture a notion of security by a picture representing a game played with adversary A. The games have one of two goals and the adversary has a range of powers.

3.2.1 Indistinguishability (IND-Security)

Indistinguishability is a measure of security whereby an adversary offers you (the challenger) two messages, and if you were to encrypt only one of them, he / she would be unable to tell **which one you have encrypted**.

Figure 3.1a shows the cryptographic game which captures this for a symmetric cryptographic scheme. m_0 and m_1 are the two messages the adversary gives you and c^* is the ciphertext you computed from one of the messages. b' is the answer the adversary gives. Note that $|m_0| = |m_1|$, since a difference in size would be a dead give away.


$b \in \{0,1\}$
$m_0, m_1 \longleftarrow$
$c^* = \mathrm{Enc}_k(m_b) \longrightarrow$
$b' \longleftarrow$
A
$b \in \{0,1\}$
$pk \longrightarrow$
$m_0, m_1 \longleftarrow$
$c^* = \mathrm{Enc}_{sk}(m_b) \longrightarrow$
$b' \longleftarrow$
A

(a) Symmetric Key Case (b) Public Key Case

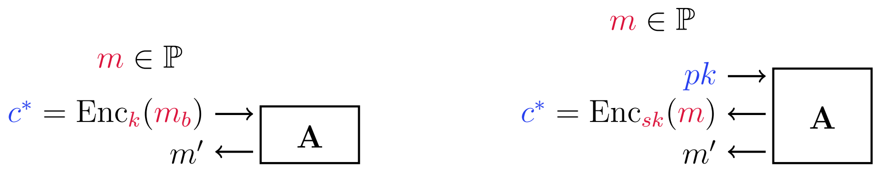
Figure 3.1: IND-Security Games

Figure 3.1b shows the game design for public key encryption. pk is the publicly available key, which all adversaries would have access to while trying to crack your message.

A cryptographic system fails a game if an adversary can win the game more often than chance (i.e. $> 50\%$ success rate).

3.2.2 One-wayness (OW-Security)

One-wayness is the property that an attacker, with only the ciphertext, **cannot decrypt the message**. The layout of the games used for this are in Figure 3.2.



(a) Symmetric Key Case (b) Public Key Case

Figure 3.2: OW-Security Games

3.2.3 Adversarial Powers

These attacks also have defined 'adversarial powers', where the adversary has access to specific oracles.

- **Passive Attack**: Has no oracles — all games above are passive attacks.
- **Chosen Plaintext Attack (CPA)**: The adversary can encrypt any mesage of his/her choosing.
- **Chosen Ciphertext Attack (CCA)**: The adversary can decrypt any message of his choosing, except he is not allowed to decrypt c^*.

We assume that if an adversary has access to a decryption oracle, then they have access to the encryption one, so CCA is an extension of CPA[1]. There is no notion of a passive attack for public key encryption because the public key (which is used to encrypt) is public, so the adversary always has an encryption oracle.

3.3 Reductions

We can make comparisons of problems by reducing one to another, thereby defining one as 'no harder' than the other. A reduction is where we can, in polynomial time, convert a problem into another one. We say **Problem A is no harder than Problem B** if we can convert Problem A into Problem B. This is written as $\boldsymbol{A \leq_P B}$.

Some crypto games are harder to beat than others. Figure 3.3 show these relationships

[1]Except in one case we will see later on concerning hybrid encryption schemes.

$$\begin{tikzcd}
\boxed{\text{IND-CCA}} \arrow[r] \arrow[d] & \boxed{\text{IND-CPA}} \arrow[r] \arrow[d] & \boxed{\text{IND-PASS}} \arrow[d] \\
\boxed{\text{OW-CCA}} \arrow[r] & \boxed{\text{OW-CPA}} \arrow[r] & \boxed{\text{OW-PASS}}
\end{tikzcd}$$

Figure 3.3: Relationships between attacks

between the attacks. An arrow ($A \rightarrow B$) means A is harder to beat, and thus more powerful, than B and thus a proof that a system meets A's notion of security, also proves it meets B's.

IND-CCA is the de-facto security definition we should accept. To pass IND-CCA, an encryption scheme must be probabilistic (i.e. encryption is a one-to-many function), and it must implement a MAC scheme to make utilising the decryption oracle on slightly modified c (because we are not allowed to use it on c) impossible.

3.4 Stream Ciphers

The idea behind a stream cipher is to replace the (possibly) huge cipher needed for the One-Time-Pad scheme by a pseudo-random sequence which is 'seeded' by a key of a more practical size. There's some confusion over whether the term 'stream cipher' refers to the algorithm generating the stream or the entire encryption scheme. The term can mean both, but the book accompanying the unit recommends you only use it for the algorithm.

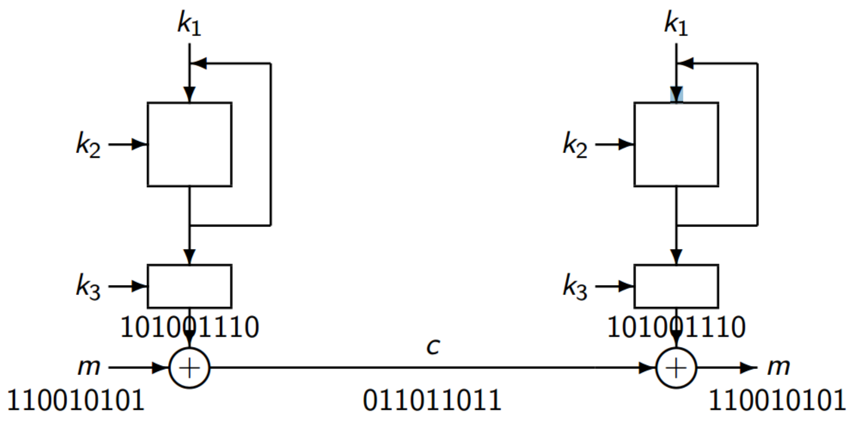


Figure 3.4: A stream cipher using a DFA where keys determine state update (k_1), initial state (k_2) and output filter (k_3)

One model of a stream cipher is a finite state machine, where the key provides the initial state, variables used to convert a keystream, k_i, to the next keystream, k_{i+1}. The ouput of the FSA is also XORed with a value taken from the key.

One source for confusion may come from that fact that the slides use k_x to mean both the key values which are input into the stream cipher, and for the keystream, which is the output of the stream cipher.

Decryption is easy as both sender and receiver use one key to generate the keystream of necessary size. This is possible because the algorithm is deterministic, which, as mentioned later, is a source of weakness.

Chapter 4

MATHEMATICS

4.1 Modular Arithmetic & Groups

4.1.1 Modular Arithmetic

Modular arithmetic is a bit odd at first, but you get used to it. It's all about remainders, really; we express modular relations in the following form:

$$X \equiv Y \underline{\boldsymbol{MOD}}\ Z$$

Where X is any old number and Z is the 'Modulus'. Y is the remainder of the integer division of X by Z, the result of an operation expressed in many languages (including C) as $X\%Z$. Simples.

When you are doing modulo maths in crypto, you only really need to remember 1 rule:

$$(a+b) \mod N \equiv \big[(a \mod N) + (b \mod N)\big] \mod N$$

I've used addition when writing that rule down, but it's the same for subtraction and multiplication. The reason for this is that addition and multiplication **group operations** on the set of modulo remainders. SAY WHAT?

This leads us nicely onto groups.

4.1.2 Groups

A mathematical group is a combination of a set and an operation, usually written $(G, *)$. We require three properties of this operation;

$$\begin{aligned}
\text{Closure: } & \forall x, y \in G \quad x * y \in G \\
\text{Associativity: } & \forall x, y, z \in G \quad (x * y) * z = x * (y * z) \\
\text{Identity: } & \exists e \in G\ st\ \forall x \in G \quad e * x = x * e = x \\
\text{Invertability: } & \forall x \in G\ \exists x^{-1} \in G\ \ st\ \ x * x^{-1} = x^{-1} * x = e
\end{aligned}$$

Most cryptographic constructions used here are defined on the group of non-zero integers modulo N under modular multiplication, defined thusly:

$$\mathbb{Z}/N\mathbb{Z}^* = \{1, 2, 3, ..., N-1\}$$
$$\forall x, y, \in \mathbb{Z}/N\mathbb{Z}^* \quad x * y = xy \mod N$$

This is actually only a group if N is prime. You should think about why 0 is not a member of this group, and then why, if N is not prime, this is not a well-defined group. We'll deal with this later when we talk about rings.

Whilst the only concrete we'll deal with here is $(\mathbb{Z}/N\mathbb{Z}, \cdot)$, other groups exist and are used in the real world, and cryptographic constructions and problems are often defined over a generic group, rather than over this specific one.

If we have the extra special benefit of commutativity in our operation, ie $x * y = y * x$, then this is an '**Abelian Group**' (sometimes just called a 'commutative group').

Groups can sometimes be defined in terms of **generators**. We say that $g \in G$ is a generator for $(G, *)$ iff we can rebuild G exclusively from g^n, with various n, where n is a natural number. This differs from the idea of a generating set which does not appear to be part of this course, but if you're interested, then we say that $S \subseteq G$ is a generating subset of G iff G can be expressed using only finite products of elements of S and their inverses.

4.1.3 Rings

In fact, we can define both the addition and multiplication operations in $\mathbb{Z}/N\mathbb{Z}$. Rather than defining two seperate groups, we combine the two into a new thing called a 'ring'. A ring in this context is a set with operations for addition and multiplication, with all the properties you might expect.

It's like an abelian group with respect to addition, because addition is closed, associative, commutative, has an identity and a guaranteed inverse. Not only that, my mentally-endowed comrades, but it is like an abelian group for multiplication too, with the added awesome that multiplication is distributive over addition.[1]

4.1.4 Multiplicative Inverse

The 'inverse' for a group member is another member of the same group that when used as the second argument for the operation gives the result of the identity for that operation.

In this case, the operation is multiplication, the multiplicative identity is 1, and our group is usually numbers Modulo N. So the inverse of one number Mod N is another number less than N, that when you multiply them together you get a number which is 1 Mod N. Still here? Still awake? Stay with me, soldier.

[1]The minimal spec for a ring is an abelian group with a second associative binary operation that is distributive over the main group operation.

4.2 Greatest Common Divisor

Say you've got two numbers. What is the biggest factor that they both share? BOOM. DONE.

Ok, well, there's probably this really handy algorithm you should know; the **Euclidean algorithm**. Invented by ~~Nigel Smart~~ Euclid bloomin' ages ago, it's a fairly simple way to efficiently figure out the GCD of 2 numbers.

- Let k be a number, starting at 0, that is incremented at each stage of this algorithm
- Let $r_{-2} = a, r_{-1} = b$, where a and b are the two numbers you are testing, and a is the larger of the two.
- A stage is this:
$$r_{k-2} = q_k \cdot r_{k-1} + r_k$$
- Where q_k is the quotient of the integer division $\frac{r_{k-2}}{r_{k-1}}$, and r_k is the remainder.
- Keep going until $r_k = 0$, at which point r_{k-1} is the GCD. If r_{k-1} is 1, then your original numbers are **co-prime** (aka **relatively prime** or **coprime**), meaning that they have no common factors.

4.3 Euler Phi Function

Also known as Euler's Totient Function. Because why the hell not. It is simply a measure of how many members of our set of remainders are **relatively prime** to N. In other words, how many members share no common factors with N. If we choose N to be prime, which we normally do, then $\phi(N) = N - 1$. Easy, right?

The other important case for cryptography is where N is made from two prime factors (p and q). In this case N's Euler Phi is more simply defined thus:

$$\phi(p \cdot q) = (p-1)(q-1)$$

Since you are undoubtedly on a roll, here is the actual formula (it's not too scary); the first is the prime factorization of N, which we need.

$$N = \prod_{i=1}^{n} p_i^{e_i} \textbf{ allowing us to use } \phi(N) = \prod_{i=1}^{n} p_i^{e_i - 1}(p_i - 1)$$

4.4 Lagrange's Theorem

Lagrange's theorem states for a group G of order (size) n then for all $a \in G$:

$$a^n = 1$$

So in a group modulus under addition, the following also holds:

$$a^{\phi(N)} \equiv 1 \mod N$$

4.4.1 Fermat's Little Theorem

Fermat's little theorem is a special case of Lagrange's theorem, and a little theorem it is. It simply states that, if p is a prime number, then for all a:

$$a^p \equiv a \mod p$$

4.5 Fields

A field is a special variety of **abelian ring**. Whereas a ring's second (multiplicative) operation doesn't need to have an inverse for every non-zero element, a field's does.

4.6 Chinese Remainder Theorem

The Chinese Remainder Theorem, often written CRT, states that given a modulus N, made up of pairwise coprime factors n_1, ..., n_r, it is possible to calculate $x = y \mod N$ by instead calculating modulo its component parts, n_1, ..., n_r and the combining those parts using some clever maths we'll get onto in a moment. This isn't particularly useful with small N, but you can imagine when your N and x are in the 2^{1024} range, and you're performing modular exponentiation it has some performance implications.

Anyway, back to the maths... The CRT states that $x = y \mod N$ can be reconstructed based on answers to each sub-modulus thus:

$$x = \sum_{i=1}^{r} a_i N_i y_i \mod N$$

Where a_i is the remainder modulo n_i, $N_i = N/n_i$, and $y_i = N_i^{-1} \mod n_i$

The significant thing to note here is that this means if we know what x is equal to modulo the component parts of N, then we know what x is equal to modulo N.[2]

4.7 One Way Functions

Firstly, lets define a function simply as something that, for items in set X, provides a mapping to items in set Y:

$$f \colon X \to Y : x \mapsto y = f(x)$$

Such a function is known as a **one way function** if, for all x, $f(x) = y$ is efficient to compute, while computing the x that maps to y is infeasible. When we say infeasible, we mean its so bloody hard it would essentially take, like, **forever**.

There is a special subset of one way functions, known as **trapdoor one way functions**. These are so named because, while reversing them naively is infeasible, given some extra **trapdoor information**, they are **easily inverted**. An example of such a

[2]And presumably vice versa, though not so sure about this.

function would be the RSA algorithm, where deducing m from c is impossible without the necessary trapdoor information, which in the case of RSA is the private key.

4.8 SQROOT Problem

The square root problem, often written SQROOT, is that it is bloody difficulty to find x where:

$$y \mod N = x * x \mod N$$

Even when y and N are known, without knowing N's factors.

Chapter 5

SYMMETRIC MODES

PRIVATE KEY CRYPTOGRAPHY

5.1 Block Cipher Modes

A **block cipher** is type of encryption based on **permutation**, where they key, $\{0,1\}^k$, defines a transposition of bits in the block to the encrypted ciphertext block, $\{0,1\}^b$. The block is a proportion of the message.

DES was the first civilian block cipher and was developed at IBM in the 1970s. When the US government adopted it, they recommended the following four ways to use it; these are now used with any block cipher.

- Electronic Code Book
- Cipher Block Chaining
- Counter
- ...

5.1.1 Electronic Code Book

This is a very simple, it **divides** the message into blocks of size b, pads the last one, and **encrypts them all individually**. This is ineffective to say the least, as evidenced in Figure 5.4. A block cipher using ECB is only OW-CPA, as Figure 5.1 says. This is terrible!

$$\begin{tikzcd}
\boxed{\text{🟥 IND-CCA}} \arrow[r] \arrow[d] & \boxed{\text{🟥 IND-CPA}} \arrow[r] \arrow[d] & \boxed{\text{🟥 IND-PASS}} \arrow[d] \\
\boxed{\text{🟥 OW-CCA}} \arrow[r] & \boxed{\textbf{🟩 OW-CPA}} \arrow[r] & \boxed{\text{🟩 OW-PASS}}
\end{tikzcd}$$

Figure 5.1: Security Models ECB passes

The weakness of this mode lays in the fact that the blocks are encrypted independently. This is the basis for the two attacks below, but it also means it's susceptible to **block**

replay. That is where an adversary edits a bit knowing how it will affect the decrypted message. If I knew you were sending me some money, and I knew the block containing the amount you were transferring, I could change that block in the hope I would end up with a larger transaction. This could be fixed with a checksum. ECB has one positive aspect though; an error in one block of the ciphertext will not propagate to other blocks during decryption or encryption.

Proving a cryptographic system passes a security model is beyond the scope of this course, but you should be able to give an intuitive reason. It passes OW-CPA because, with only an encryption oracle, you would have to brute-force every possible message to match it with the ciphertext. We can always make b large enough so that this is not possible[1]. Note that all the proofs for block cipher modes assume we are using a 'perfect block cipher'.

OW-CCA Example: We can win a OW-CCA game by kind of cheating in the following way. If we have a decryption oracle, we can decrypt anything that is not the message. Since the blocks are independent, we can simply split the message in half and decrypt both individually and then concatenate the result.

IND-PASS Example: In an indistinguishability game we decide the two messages sent to the oracle (of which one will be encrypted). Again we use the fact that the blocks are encrypted independently, and give one of the messages as a bit string concatenated onto itself, $m_0 = b_1||b_1$. If the ciphertext can be split into two equal bit strings in the same way, then m_0 was encrypted, else m_1.

5.1.2 Cipher Block Chaining

Cipher Block Chaining removes a lot of the problems found in ECB by XORing each block with the previously encrypted block — incorporating a dependence on the previous block. The first block of the ciphertext is called the **Initalisation Vector** (IV), and is what the first block of the message is XOR-ed with.

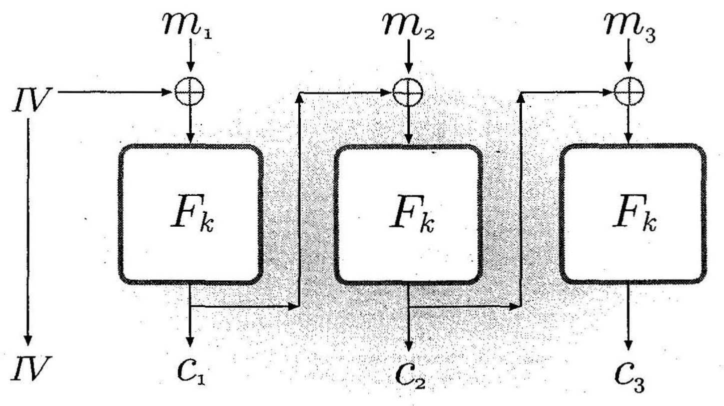


Figure 5.2: Diagram of CBC encrypting

[1]This might be possible if you knew the context of the message, if you understand what is being sent and the domain of possible values of a block is relatively small. This is simply poor implementation of the encryption, however, and we don't worry about that.

More formally:

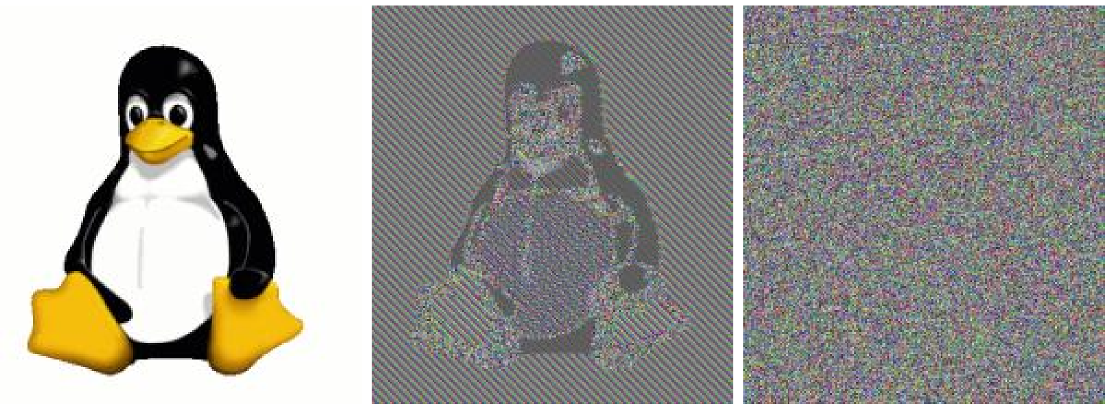
Encryption:
$c_0 = IV$
$c_1 = Enc_k(m_1 \oplus IV)$
$c_i = Enc_k(m_i \oplus c_{i-1})$ for $i > 1$

Decryption
$IV = c_0$
$m_1 = Dec_k(c_1) \oplus IV$
$m_i = Dec_k(c_i) \oplus c_{i-1}$ for $i > 1$

$$\begin{tikzcd}
\boxed{\text{IND-CCA}} \arrow[r] \arrow[d] & \boxed{\textbf{IND-CPA}} \arrow[r] \arrow[d] & \boxed{\text{IND-PASS}} \arrow[d] \\
\boxed{\text{OW-CCA}} \arrow[r] & \boxed{\text{OW-CPA}} \arrow[r] & \boxed{\text{OW-PASS}}
\end{tikzcd}$$

Figure 5.3: Security of Models CBC

Note that IV is sent unencrypted, because it is needed for decryption. This can seem pointless, but if it is generated randomly every time, then it means that encryption is probabilistic. CBC is OW-CPA and IND-CPA but not OW-CCA or IND-CCA.



(a) Original Image (b) Encrypted using ECB (c) Encrypted using CBC

Figure 5.4: Bitmap image of Tux being encrypted using ECB and CBC.

With a decryption oracle, CBC will fail because of the same trick used before — we can ask to oracle to decrypt the ciphertext with extra blocks on the end.

5.1.3 Counter Mode

Counter mode basically uses a stream cipher to generate a pseudo-random bit-string which can be one-time-padded with message. As Figure 5.5 shows, the algorithm generates a IV ($IV \leftarrow \{0,1\}^n$) and increments this by one for every block. The $IV + n$ values are then put through the block cipher to generate the bit-string. In all the material, the IV is refereed to as *ctr* for this algorithm.

Formally:

Encryption:
$c_0 = IV$
$c_i = Enc_k(IV + i) \oplus m_i$

Decryption:
$IV = c_0$
$m_i = Enc_k(IV + i) \oplus c_i$

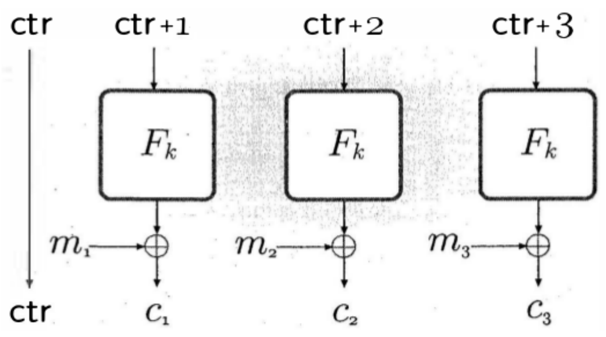


Figure 5.5: Diagram of CTR mode

CTR mode passes the same security modes as CBC (Figure 5.6). But it has one positive over CBC: decrypting a block is not dependent on the decryption of the previous block, meaning that we can parallelise decryption (and encryption for that matter).

$$\begin{tikzcd}
\text{IND-CCA} \arrow[r] \arrow[d] & \textbf{IND-CPA} \arrow[r] \arrow[d] & \text{IND-PASS} \arrow[d] \\
\text{OW-CCA} \arrow[r] & \text{OW-CPA} \arrow[r] & \text{OW-PASS}
\end{tikzcd}$$

Figure 5.6: Security of Models CTR

5.2 S-Box & Analysis

Until now, we haven't talked about what happens in the *Enc* function. In there, there is a mixture of **substition** of characters for other characters[2] and **permutations**, where bits (or larger groupings of bits) change position. When there is this mixture of subtition and permutation, we give it the imaginative name of a 'substitution-permutation network'.

One method an attacker might use is called a **known-plaintext attack**, where they have the *plaintext* and *ciphertext* and try to work the key out from these. This means we need a large number of keys, so that an exhaustive keysearch is not possible. We also have to make sure that the block size is large enough, since we could store known block decryptions and decipher a large part of the ciphertext through this (**text dictionary attack**).

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	E	7

Table 5.1: Example of a look-up table for an S-Box

An **S-Box** is a mapping of bit-strings to other bit-strings, theses do not have to be the same length, but if they are not, the output is almost always of a shorter length. Table 5.1

[2]Think of Caesar's cipher where a letter is moved n places in the alphabet, where n is defined in the key

shows an example of an S-Box which maps 4-bit strings to 4-bit strings. We want to make the substitution as **non-linear** as possible. Formally, it would be linear if Equation 5.1 was true, or at least was true was large probability.

5.2.1 Differential Analysis

At the heart of a number of the techniques used to successfully beat crypto games in Section 5.1, was that we get around the constraint of not being allowed to send the message or ciphertext to an oracle by changing it into something we know will still give us something useful. With a linear S-Box, we (as an attacker) would be able to transform the message/ciphertext and then perform another transformation on the oracles response.

$$\left.\begin{matrix} m_1 \to c_1 \\ m_2 \to c_2 \end{matrix}\right\} \Rightarrow (m_1 + m_2) \to (c_1 + c_2) \tag{5.1}$$

To find patterns that we can exploit to do this, we perform **differential analysis** on the S-Box. Differential analysis *tabulate[s] specific differences in the input that lead to specific differences in the output with probability greater than would be expected for a random permutation*. This would tell us whether adding 5 to the input will always give us an output 10 too large.

To do this, we make a table with all the inputs against all the outputs. Then we take **every pair of inputs**, (s_1, s_2), calculate there difference, $\Delta s = s_1 \oplus s_2$, and the the difference between the outputs of those inputs, $\Delta t = \mathbb{S}(s_1) \oplus \mathbb{S}(s_2)$. Figure 5.7 gives an example of this. For each pair, we add one to the value at $(\Delta s, \Delta t)$, so that cell (s, t) gives us the number of times a difference of s in the input caused a difference in t in the output.

$$\begin{array}{lll} s_1 = 3 = 0011_2 & t_1 = \mathbb{S}(s_1) = 1 = 0001_2 & \Delta s = s_1 \oplus s_2 = 5 = 0101_2 \\ s_2 = 6 = 0110_2 & t_2 = \mathbb{S}(s_2) = B = 1011_2 & \Delta t = t_1 \oplus t_2 = A = 1010_2 \end{array}$$

Figure 5.7: Example of differential analysis for the pair of inputs $(3, 6)$

Although its unlikely that the table will show all differences in the input of s lead to an output difference of t, even a *better than chance* difference gives a cryptanalyst a much smaller space for them them brute force. A perfect S-Box would have a 1 in every cell, but this is *impossible*. We can measure an S-Boxes quality by how close to a **uniform distribution** it has in this table.

5.2.2 Linear Analysis

Linear analysis consists of two parts. The first is to **construct linear equations** describing relationships between the *plaintext*, *ciphertext* and *key bits* that have a high bias; that is, whose probability of being true is close to 0 or 1. The second is to use these linear equations in conjunction with known plaintext-ciphertext pairs to derive key bits.

The notes write a linear equation for an S-Box has n input bits X_i and m-output bits Y_i, as

$$L_{I,J} = (X_{i_1} \oplus \cdots \oplus X_{i_2}) \oplus (Y_{i_1} \oplus \cdots \oplus Y_{i_2}) \tag{5.2}$$

$$\begin{aligned} \text{where, } I &= \{i_1, \cdots, i_u\} \subset \{1, \cdots, n\} \\ J &= \{j_1, \cdots, j_u\} \subset \{1, \cdots, n\} \end{aligned}$$

because they apparently don't want you to understand it. It just means that L is a function taking bits from specific positions from the input and output and XORing them all. Eg, $L_1 = X_1 \oplus X_3 \oplus X_4 \oplus Y_2$.

For each linear equation, L_i, we calculate the probability that for a random input, X, the function will return 0: $p = Pr[L_i = 0]$. To obtain the bias, we just subtract -0.5 from this probability, making the bias in a range of -0.5 to 0.5.

If function has a large enough bias (either positive or negative), then we can approximate the whole cipher as a linear function. Using this method we can prove that an S-Box is non-linear.

5.3 AES — Rijndael

Some ciphers involve repeating a weaker 'round function' to make a stronger encryption. Round functions will output bit-strings the same size as the input, and they must be a revertible one-to-one function, so that we can decrypt the ciphertext. The notation usually uses r for the number of rounds, n for the block size and s for the key size. Each round uses a different key, and these sub-keys are all derived from the main key. A block is usually represented as a matrix holding all the bytes. Each state of the encryption is applied to the 'state matrix', s.

AES is an encryption scheme which uses round functions. It works on 128-bit blocks, uses keys of 128/192/256-bits and works over 10/12/14 rounds. Each round consists of the following operations:

Byte Substitution
Just apply the S-Box to the block.

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,0} & s_{2,1} & s_{2,2} \end{pmatrix} \to \begin{pmatrix} \mathbb{S}(s_{0,0}) & \mathbb{S}(s_{0,1}) & \mathbb{S}(s_{0,2}) \\ \mathbb{S}(s_{1,0}) & \mathbb{S}(s_{1,1}) & \mathbb{S}(s_{1,2}) \\ \mathbb{S}(s_{2,0}) & \mathbb{S}(s_{2,1}) & \mathbb{S}(s_{2,2}) \end{pmatrix}$$

Shift Rows Shift a row, so that message is diffused over the columns.

$$\begin{pmatrix} \color{red}{s_{0,0}} & \color{green}{s_{0,1}} & \color{blue}{s_{0,2}} \\ \color{red}{s_{1,0}} & \color{green}{s_{1,1}} & \color{blue}{s_{1,2}} \\ \color{red}{s_{2,0}} & \color{green}{s_{2,1}} & \color{blue}{s_{2,2}} \end{pmatrix} \to \begin{pmatrix} \color{red}{s_{0,0}} & \color{green}{s_{0,1}} & \color{blue}{s_{0,2}} \\ \color{green}{s_{1,1}} & \color{blue}{s_{1,2}} & \color{red}{s_{1,0}} \\ \color{red}{s_{2,0}} & \color{green}{s_{2,1}} & \color{blue}{s_{2,2}} \end{pmatrix}$$

Mix Column
Can choose a matrix to multiply by. Diffuses over rows. [3]

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,0} & s_{2,1} & s_{2,2} \end{pmatrix} \to \begin{pmatrix} 2 & 3 & 1 \\ 1 & 1 & 4 \\ 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,0} & s_{2,1} & s_{2,2} \end{pmatrix}$$

Round Key Addition
XOR with the round key.

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,0} & s_{2,1} & s_{2,2} \end{pmatrix} \to \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} \\ k_{1,0} & k_{1,1} & k_{1,2} \\ k_{2,0} & k_{2,1} & k_{2,2} \end{pmatrix} \oplus \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,0} & s_{2,1} & s_{2,2} \end{pmatrix}$$

It's possible that you need to know they are put together in AES, so you can see the pseudo-code in Algorithm 1. Basically, every round the sub-key is added, then the S-Box is applied, then the rows are shifted, then the columns are mixed. In the last round the columns aren't mixed though.

```
AddRoundKey($S$,$K_0$]);
for $i \leftarrow 1$ to 9 do
    SubBytes($S$);
    ShiftRows($S$);
    MixColumns($S$);
    AddRoundKey($S$,$K_i$);
end
SubBytes($S$);
ShiftRows($S$);
```

Algorithm 1: AES

Clearly we can't just use any S-Box, they have to not be susceptible to differential and linear analysis. The S-Box used is known as the Rijndael S-Box. It can be broken down into two sections:

a) **Multiplicative Inverse**: Remember the multiplicative inverse, x^{-1}, of x is the number in the finite space for which $x \cdot x^{-1} = 1$. So in this context, we want $x \cdot x^{-1} \mod 2^8 = 1$. It must equal 1 because 1 is the identity number for multiplication ($x \cdot I = x$). We count zeros inverse as zero.

b) **Linear map** over $\mathbb{F}_2$, which just means its XORed with a bit-string.

[3] The aim is to diffuse the message as much as possible, so a maximal distance algorithm is used to find the matrix which will do this the most — you don't need to worry about that though.

Put them together, and you make the equation of the form $\mathbb{S}(x) = A^{-1} + c$. And it turns out, because of the modular arithmetic[4], it can actually be placed in the form $\mathbb{S}(x) = A'x \oplus c'$. The actual formula is below, where y_n is the n^{th}-bit in the output of the S-Box.

$$\begin{pmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{5.3}$$

Rijndael's Key Schedule
The key schedule is the mechanism used to 'expand' the 128-bit key into 10 128-bit keys[5]. The first round's key is simply the original key. For the rest, the process is as follows (for the i'th round):

1. Copy the $i - 1$'th round's output into a temporary variable t.
2. Bitwise rotate t left by 1 byte.
3. Apply the S-Box to each byte of t.
4. XOR the **first byte only**, with the output of $rcon[i]$.

So, what's Rcon I hear you cry? Never fear, I shall explain, though Martijn's slide gloss over this so if you're struggling to remember things as it is **just remember the above**. The Rcon function is simply defined as the following operation in the Rijndael finite field:

$$rcon(i) = x^{i-1}$$

So this is in effect, another S-Box! Just as with the AES S-Box, we simply use it like a lookup table, hence the square bracket notation in the algorithm.

5.3.1 Why?

Basically, the Multiplicative Inverse adds a non-linear component (the only component in the S-Box) and the linear map removes stationary points (when $\mathbb{S}(n) = n$).

Remember differential analysis looks at how many times a difference in input causes a difference in output. So, we want to make a table that shows how many times $\Delta y = f(x) \oplus f(x + \Delta x)$ holds for the pair $(\Delta x, \Delta y)$, in the finite field $\mathbb{F}_{(2^8)}$.

Multiplicative Inverse: Lets look at a function $f(x) = 1/x$. This means we want to find

$$\Delta y = \frac{1}{x} \oplus \frac{1}{x + \Delta x} \tag{5.4}$$

So we can bring up the denominators, we we say that $x \neq 0$ and $x + \Delta x \neq 0$.

[4] I think...
[5] The number of rounds is linked t the key length, it is only 10 rounds for 128-bit keys.

5.4 Message Authentication

5.4.1 Why Do This

Encrypting data provides confidentiality (remember the three goals), but does not provide authenticity or integrity without additional sauce. This is the additional sauce that provides **integrity**. It comes with two flavours: **MDC** (Manipulation Detection Codes) and **MAC** (Message Authentication Codes). We will mainly worry about **MAC** stuff. The reason we use these at all is that they are the secret ingredient in getting from IND-CPA to the mythical, coveted IND-CCA level of security.

5.4.2 Message Authentication Codes, you say?

MAC codes are the result of hashing the message, using a hash function that takes a key. You would then typically send the MAC concatenated at the end of the message.

$$\text{MAC} = h_k(m)$$

The hash function, as per Kerckhoff's principle, is publicly described. Given k and m, computing $h_k(m)$ should be easy. Given only m, computing a correct hash should be very difficult, even if several message-to-hash pairs are already known.

A hash function is simply a surjective mapping from arbitrarily long strings to fixed length hash strings.

5.4.3 Security Model

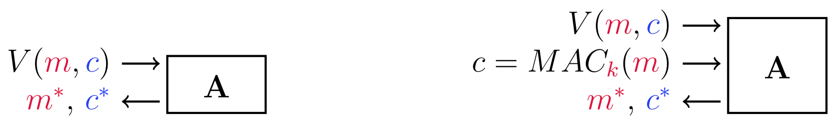
Our security models are simple:

- **EF-PASS**: The passive attack where the bad guy can generate a valid MAC for any message, gibberish or otherwise. **Existential** forgery is where the message may just be random rubbish, **Selective** forgery is creating a MAC for a specific message.
- **EF-CMA**: As above, but the douche of an adversary has an oracle that can perform MAC generation on other messages of his choice (but not the target message). This is the **Chosen Message Attack**

We will now look at two games for Existential Forgery; as a MAC may be probabilistic, we define two algorithms for these games:

- **MAC**$_k(m)$: Generates the MAC for message $\underline{\text{m}}$.
- $V_k(m, c)$: A boolean-returning verification function. True is good. False is bad. Get with the program, kids. Also this may not simply be a case of recomputing the MAC; if the algorithm that generates it is probabilistic, we need other methods of checking correctness.

It is a given that our opponent has access to a Verification Oracle (otherwise how will they know if they have cracked it?).



(a) EF-PASS: Passive Attack (b) EF-CMA: Chosen Message Attack

Figure 5.8: MAC security games

There exists a 'Strong Forgery' variant of **EF-CMA** called, unsuprisingly, **SF-CMA**, which changes the restriction on the MAC oracle to be that whilst m^* can be passed to the oracle, c^* must not have been returned.

Assuming we can create a MAC function that achieves this, we can now make INC-CCA symmetric encryption schemes! Hooray!

There exists a 'Strong Forgery' variant of **EF-CMA** called, unsuprisingly, **SF-CMA**, which changes the restriction on the MAC oracle to be that whilst m^* can be passed to the oracle, c^* must not have been returned.

Assuming we can create a MAC function that achieves this, we can now make INC-CCA symmetric encryption schemes! Hooray!

5.4.4 IND-CCA Here We Come

The reason that CBC and CTR modes, whilst groovy, were not **IND-CCA** was that an attacker could look at our ciphertext and construct a related one which could be decrypted through their oracle, giving them a related plaintext.

MAC prevents them from doing that! Sweet! So to make an **IND-CCA** secure scheme you will need:

- One **IND-CPA** secure symmetric cipher E.
- One SECURE[6] **MAC** function MAC.
- One hybrid key consisting of k_0 and k_1 which are the keys for E and MAC respectively.

We can then construct encryption and decryption thusly:

This obviously involves the ciphertext being expanded (more than it might be already) because it includes a MAC.

With the MAC allowing us to validate a ciphertext as being 'authentic', this type of scheme is often called 'authenticated encryption'. It is important to realise the **very important** distinction between this idea, which is that a ciphertext is authentic, and the concept that a ciphertext came from where it was supposed to, which is part of the regular cryptographic definition of 'Authenticity'.

[6]i.e. Cannot produce a valid **MAC** without access to the correct key

Encrypt

1. Split k into k_0 and k_1
2. $c_0 = E_{k_0}(m)$
3. $c_1 = MAC_{k_1}(c_0)$
4. And voila: $c = (c_0, c_1)$

Decrypt

1. Split k into k_0 and k_1
2. Split c into c_0 and c_1
3. $c_1^* = MAC_{k_1}(c_0)$
4. If $c_1^* \neq c_1$ then ABORT and return $\perp$
5. Else return m where $m = D_{k_0}(c_0)$

Figure 5.9: How To Make an IND-CCA Scheme

5.4.5 How To Make A MAC

There are a few types of MAC schemes, from MACs that are custom made for that specific encryption scheme to MACs that are derived from MDCs (Manipulation Detection Codes).

The most popular, and exceedingly examinable, variants that we look at are all based on CBC-mode block ciphers. They are covered by various international standards[7] and are very widely used. We shall refer to schemes like this under the title of CBC-MAC. Like this...

5.4.6 CBC-MAC

CBC-MAC is pretty much a straightforward application of a block cipher (like DES or AES). We just have to add the MAC on after padding the ciphertext.

Given our cipher, which operates on blocks of $\underline{b}$ bits, we can make a MAC really simply; assuming we have $\underline{q}$ data blocks in our message, $m_1, m_2 ... m_q$, we would do as follows:

1. Set initial intermediate variables: $I_1 = m_1, O_1 = e_k(I_1)$
2. For $i = 2, 3 ... q$:
   - $I_i = m_i \oplus O_{i-1}$
   - $O_i = e_k(I_i)$
3. Do any post-processing you want on O_q.
4. Truncate if necessary to m bits and serve piping hot.

In terms of padding the ciphertext, there are 3 schemes suggested in those groovy groovy standards. All of these pad to a whole number of blocks.

- Just add zeroes. Easy, right?

[7] Gotta love standards. Thrilling stuff

- Add a single 1, then trail out with zeroes.
- Pad with zeroes until you get a whole number of blocks, then add a block containing the length of the unpadded message.

In terms of optional post-processing, there are two specified methods:

- Choose another key, k_1, and replace O_q with $e_k(d_{k_1}(O_q))$
- Choose another key, k_1, and replace O_q with $e_{k_1}(O_q)$

Either one of those can make it much harder to do a brute force search for k. Which is a good thing, young padawan. Oh yes.

5.4.7 Hashing

Like we said, hashing in this case is just an efficient function mapping arbitrarily long binary strings to fixed length binary strings. Simples.

Incidentally, hash functions are also known as MDCs (manipulation detection codes). To use the for this purpose, you hash your message, concatenate it to the message and then encrypt that. The problem with a concatenate-then-encrypt scheme under CBC is that an attacker can create a message that consists of the message they wish to send, a hash for it, and then some other random guff, and encrypt that (this is a CPA method). If the attack then truncates the resulting ciphertext they get their valid target ciphertext, complete with hash.

While earlier we talked about hashing with a key, in practice hash functions don't have a key. It is often simpler to consider a family of functions, and require three conditions of them:

1. Preimage Resistance: given $c = h(m)$, hard to find another m' such that $h(m') = c = h(m)$
2. 2nd Preimage Resistance: The same as preimage resistance, except that we are also given m.
3. Collision Resistance: hard to find $m, m' \neq m$ such that $h(m) = h(m')$

Typical practical choices for a cryptographic hash are the SHA family, or the RIPEMD family.

5.4.8 From Hash To MAC

Given a hash function, how can we bung a key in there sensibly? There are the simple prefix, suffix and envelope methods, though these are not secure.

- With prefixing it is possible to calculate $MAC_k(m||m')$ without knowing the key. Simply split your intended message in half, and you're good to go. Exact method not shown in notes.

- Suffixing suffers from a weakness that makes it easier to find collisions in the hash function; this can be done offline, as well, so you do not need to send lots of queries to the target. Exact methodology not disclosed in notes.
- Envelope method with padding: The reason this isn't secure isn't given in the notes, so chillax man!

It is better to use HMAC (keyed-**H**ash **M**essage **A**uthentication **C**ode):

$$HMAC_k(m) = h(k||p_1||h(k||p_2||m))$$

Where p_1 and p_2 are fixed strings used to pad k to a full block.

You can use both MDC and MAC for data integrity, with or without confidentiality.

Without confidentiality:

- **MAC**: compute $MAC_k(m)$ and send $m||MAC_k(m)$
- **MDC**: Send $h(m)$ over a seperate, authenticated channel.

With confidentiality:

- **MAC**: need two different keys, k_1 and k_2
  - k_1 is for computing $c = e_{k_1}(m)$
  - k_2 if for computing $MAC_{k_2}(c)$, which you append to c before sending that combined string.
- **MDC**: we only need one key k for encryption, where we send $c = e_k(m||h(m))$, but as we discussed earlier this can be compromised by a man in the middle attack.

Chapter 6

ASYMMETRIC MODES

PUBLIC KEY CRYPTOGRAPHY

6.1 Introduction

The basic concept of public key cryptography is the idea of a box with two keys, a public key, and a private key. The **public key** allows anyone to **leave a message**. They cannot however read each others messages. The **private key** by contrast, allows its owner to **read any message left**. It is this asymmetry that gives the scheme its name.

Or to put it more concretely:

Message + Public Key = Ciphertext
Ciphertext + Private Key = Message

Throughout this chapter, you will see a lot of colours. If you see red, the highlighted symbols are intended to be kept private, in that they are hidden from the adversary. If you see blue, the highlighted symbols are public, and visible to anyone and everyone. Whether "private" symbols are actually private in practice is often another matter entirely.

6.2 Vanilla RSA

6.2.1 Definition

RSA is actually very simple. Lets start with the encryption and decryption definitions, and work back from there.

$$c = m^e \bmod N$$

$$m = c^d \bmod N$$

Simple right? e, d, and N are the only things we need. So now lets cover just what these are.

Symbol	Maths[1]	Notes
N	$N = p * q$	Where p and q are two massive **prime** numbers.
e	$\gcd(e, \phi(N)) = 1$	A randomly chosen integer, where $1 < e < \phi(N)$.
d	$e * d = 1 \mod \phi(N)$	Computed using the XGCD algorithm.

It is worth proving that this system works, namely that decrpyting a ciphertext will always result in the original message.

Let $c = enc(m)$, and let $x = dec(c)$. We intend to prove that $x \cong m \mod N$. The system is obviously correct (and obviously insecure) in the case where $c = m = 0$, so suppose this is not the case. We have that

$$x \cong m^{ed} \mod N$$
$$x \cong m^{ed} \mod p$$
$$x \cong m^{ed} \mod q$$

Consider $x \cong m^e d \mod p$. Since $m \neq 0$, we can invoke Fermat's Little Theorem to conclude that

$$m^{ed} \cong m^{ed \mod (p-1)} \mod p$$

Since $ed \cong 1 \mod (p-1)(q-1)$, we have that, for some integer k, $ed = 1 + k(p-1)(q-1)$, hence $ed \cong 1 \mod (p-1)$. Therefore

$$m^{ed \mod (p-1)} \cong m^1 \mod p$$

So $x \cong m \mod p$. Nearly identical reasoning allows us to conclude that $x \cong m \mod q$. By the Chinese Remainder Theorem, it must therefore be the case that $x \cong m \mod N$, and hence

$$dec(enc(m)) \cong m \mod N$$

QED

6.2.2 Security

Vanilla RSA is **OW-CPA** under the assumption that the RSA problem is hard. If we could easily break vanilla RSA with a **OW-CPA**, then we could use this attack to easily solve the RSA problem. Since we assumed that we cannot easily solve the RSA problem, it must be the case that we cannot mount a **OW-CPA** attack against vanilla RSA.

It is however **not IND-CPA** secure. This is pretty obvious; the encryption function is deterministic, so when given a ciphertext from a set of two messages, we can simply encrypt one message from the set, and know that if it does not match the ciphertext we received, we must have been sent the other message.

An encryption scheme is considered **malleable** if given c_1, the ciphertext of m_1, we can compute another valid ciphertext c', from a message mathematically related to m_1.

[1]Any unfamiliar symbols or functions are described in the Mathematics section.

This is indeed the case with RSA, for instance given two ciphertexts c_1 and c_2, we could compute:

$$c_3 = c_1 * c_2 \mod N = (m_1 * m_2)^e \mod N$$

Without ever knowing m_1 or m_2! This is not a good thing.[2] Why not? Because if you encrypt the number 100, without knowing that, or the private key, I can create a valid ciphertext for the number 200, simply by encrypting $m = 2$ and multiplying our ciphertexts together, modulo N.

Due to this malleability, vanilla RSA is **not OW-CCA** secure. Recall in a CCA we are allowed to decrypt any ciphertext *besides the target ciphertext*. However, we now know how to manipulate the ciphertext while leaving the original message relatively unharmed. As such, to recover m, we simply compute $c' = c * 2^e \mod N$, then decrypt c', giving $m' = m * 2$. Recovering m is now trivial.

6.2.3 Attacking RSA

Indirect Factoring

Given any private key, d, it is possible to factor the N associated with it. In its self this isn't particularly interesting; if you have the private key, does it really matter if you can use it to break the public key?

The interesting bit comes from the fact that the same process can be used to factor N with $\phi(N)$.

$$p + q = N + 1 - \phi(N)$$

Shared Modulus

Shared modulus attacks are a specific variant of Man-in-the-Middle attacks, where several clients are communicating with a single server Each of these clients use the same modulus in their keys, though they use different values for p and q.

This attack requires the same plaintext to be sent to 2 different clients. We can then do some mathsy stuff after intercepting both ciphertexts to recover the message. This attack can be avoided by never sending the same plaintext to any two clients.

Small Exponent

This is a similar attack, but instead of requiring a shared modulus it requires the public key exponents to be 'small'. 'Small', in this context, is quite a vague term, and I don't imagine they'll ask us to pick a number.

Essentially, the adversary waits for the server to send 3 clients (i, j and k) the same message, and intercepts all three ciphertexts. Because of the voodoo of the CRT, the

[2]Unless you do Applied Security, in which case this was what you used to perform the RSA-OAEP attack successfully.

adversary can do this:

$$\begin{aligned} x &\equiv C_i \mod (N_i) \\ x &\equiv C_j \mod (N_j) \\ x &\equiv C_k \mod (N_k) \end{aligned}$$

And derive this:

$$x \equiv M^3 \mod (N_i \cdot N_j \cdot N_k)$$

Handily, because a message is always less than a modulus, this means we simply calculate the cube root of x (which will be an integer) to get the original message! BOOM!

6.3 Signatures

6.3.1 What Is A Signature Anyway?

A signature in this context is a way of providing Authenticity for a message. Where a MAC can validate the integrity of a message, a signature can validate who sent it. You generate a signature from a private key, and check it against a public key.

E.g. If Alice wants to prove she sent Bob a message, she generates a signature using her private key, and then Bob can check the resulting signature against Alice's public key.

Basically, a signature is another ciphertext, with a different key. Both decryptions have to be correct for the message to be considered authentic.

6.3.2 So How Do You Sign Stuff?

A signature scheme uses three algorithms; a key generation algorithm, a signing algorithm and a verification algorithm.

- **Key Generation**: Returns two keys, a public and private one. The public one is the verification key, which I'll call vk because that's what the notes have and I'm original like that. The private one is the signing key which, again, I'll call sk because the notes...never mind.
- **Signing**: Takes sk and the message and creates a signature s.
- **Verification**: Takes s and vk and checks if it all works out.

6.3.3 Vanilla RSA Signature Scheme

The signature scheme presented in the course is called **vanilla RSA**. It is, happily, fairly straightforward, and covered in a previous chapter. To use vanilla RSA as a signature scheme isn't too different from how it operates as a cipher, so we will just cover the differences.

Key Generation: Identical to RSA, save for what information we keep and thus *use* as the keys. In the RSA signature scheme, you have a public key consisting of N and e, and a private key consisting of d, p and q.

Signing: Really simple. $s = m^d \mod N$. Then you just send s along with c.

Verification: Also pretty simple. You decrypt c into m as normal, and then test if $m = s^e \mod N$. If it is, then the message is authentic.

6.3.4 Hash-Then-Sign Signatures

'Hash then sign' signatures are pretty much what they say on the tin. It's not a scheme, rather a description for a collection of signature schemes. The process is identical to a regular signature scheme, except we replace m with $H(m)$ in the signing and verifying steps.

If H is collision resistant, and the original *(Kg, Sign, Vrfy)* scheme was EUF-CMA[3], then the new scheme is EUF-CMA.

6.3.5 Full Domain Hash Signatures

This scheme is an application of the 'hash then sign' style signature scheme. The signing process is to perform the **inverse** of a trapdoor function on $H(m)$, for which sk is required. Verification is then achieved by applying the forwards trapdoor function to the signature, for which pk is required, and comparing the output with the actual hash of the message.

6.4 Hybrid Encryption

A hybrid scheme is one which makes use of both symmetric and asymmetric encryption schemes. Generally, an asymmetric system is used to transmit a symmetric key which is then used to send messages.

6.4.1 Key Encapsulation Mechanism - KEM

Key encapsulation refers to how we use an asymmetric encryption (or public key) scheme to send the key used to send data to a recipient securely. The formal definition is as follows:

$$KEM(pk) = (k, c)$$
$$KEM^{-1}(c, sk) = k \quad \text{if } c \text{ is an encapsulation of } k\text{, else } \perp.$$

[3]What is this? Not in other slides. Suffice to say it means 'secure' perhaps.

6.4.2 Data Encapsulation Mechanism - DEM

Data encapsulation refers to how we use a symmetric encryption scheme to send data to a recipient securely. They are generally based on a block cipher since this allows the data to be of variable length. The formal definition is as follows:

$$\begin{aligned} DEM(m, k) &= c \quad \text{where } k \text{ is a key for the symmetric key function.} \\ DEM^{-1}(c, k) &= m \quad \text{if decryption is successful, else } \perp. \end{aligned}$$

6.5 Padding Schemes

6.5.1 Introduction

A padding scheme in its simplest form is a system to ensure that when our block size does not exactly divide our message, we do not lose data, or gain data that was not in our message (i.e. we are able to distinguish padding from our original message body).

A padding scheme generally uses a block at either the beginning or start of the message to specify the length of the message body, potentially along with additional information, such as a hash to verify message authenticity.

6.5.2 OAEP

Optical Asymmetric Encryption Padding is one such padding scheme. *Incredibly simplified*[4] the scheme pads a message with a — generally SHA1 — hash of the message body. If after decryption the hash is not correct for this message body, we know the message is invalid. OAEP goes above and beyond this, but that's the gist of it.

When used with RSA, OAEP gives a scheme which is **IND-CCA** secure.

6.6 Rabin

Rabin encryption is a public key cryptography scheme that is provably more secure than RSA. Its security is based on the difficulty of the SQROOT problem, which is provably equal in difficulty to the FACTORING problem. By contrast, the RSA problem, while assumed to be hard, has not been proven to be hard.

Our private key is made up of two components, p and q, two similarly sized prime numbers where $p = q = 3 \mod 4$.[5]

Our public key is made up of two components as well, $N = p * q$ and B. B is a randomly chosen number between 0 and N

An interesting property of the Rabin scheme, as seen in Figure 6.2, the private key is not strictly speaking needed for decryption, however in reality this is a case of the

[4]Seriously this is so simplified it is only for ones intuitive understanding of why it is effective
[5]This just makes extracting roots fast, there is no cryptographic need for this.

$$c = m * (m + B) \mod N$$

Figure 6.1: Rabin Encryption Algorithm

$$m = \sqrt{\frac{B^2}{4} + c - \frac{B}{2}} \mod N$$

Figure 6.2: Rabin Decryption Algorithm

SQROOT problem; even knowing the contents of the square root, finding the square root is not a trivial task.

This is where the private key comes in. Lets focus on the portion of the equation that is hard to solve:

$$t = \frac{B^2}{4} + c$$

We instead now solve $\sqrt{t} = \pm x \mod p$ and $\sqrt{t} = \pm y \mod q$. The final step is to make use of the CRT to solve modulo N. Fortunately, the actual method beyond this point is outside the scope of the unit, so breathe a sigh of relief and move on to...

6.7 ElGamal

ElGamal is an encryption scheme built around the Diffie-Hellman problems.

6.7.1 The Diffie-Hellman Problems

There are, in fact, two Diffie-Hellman problems, a computational one and a decisional one. Let g be a generator for some group, and let x, y and z be integers. In the Computational Diffie-Hellman problem, you are given g, g^x and g^y, and must compute g^{xy}. For the decisional diffie-hellman problem, you are given g, g^x, g^y and g^z, and are required to decide if $z = xy$. ElGamal is an encryption scheme built on the assumption that both of these problems are hard.

6.7.2 The Algorithms

Let p be some large prime, usually around 1024-bits. Let q be another prime, usually around 160-bits, such that q divides $p - 1$. Compute g, a generator of a multiplicative group of order q – usually the group of positive integers modulo q under multiplication. (p, q, g) are the public domain parameters of the ElGamal scheme, and are part of the public key. g can be computed by a fairly simple Las Vegas algorithm; choose r at random, and let

$$g \cong r^{(p-1)q^{-1}} \mod p$$

If $g = 1$, pick another r and do it again. If not, g is our generator.

Now let $1 \leqslant x < q$ be some random integer, and let

$$h \cong g^x \mod p$$

h is the ElGamal public key, and x is the private key.

To encrypt some message, m, pick some random integer $1 \leqslant k < q$, and compute

$$\begin{aligned} c_1 &\cong g^k \mod p \\ c_2 &\cong mh^k \mod p \end{aligned}$$

And send $c = (c_1, c_2)$.

To decrypt some ciphertext, (c_1, c_2), compute

$$m \cong {c_1}^{-x} c_2 \mod p$$

The proof that this decryption works is somewhat simpler than it was for RSA. Expand all ciphertexts to their definitions to see that $c_1 \cong g^k$ and $c_2 \cong mg^{xk}$, and note that

$$\begin{aligned} {c_1}^{-x} c_2 &\cong (g^k)^{-x} m g^{xk} \mod p \\ &\cong g^{xk} g^{-xk} m \mod p \\ &\cong m \mod p \end{aligned}$$

So decryption inverts encryption.

6.7.3 Security

ElGamal's proofs of security are a little more difficult than they were with RSA, so pay attention. They all rely on assumptions about the Diffie-Hellman problems.

ElGamal is OW-CPA secure under the assumption that the Computational Diffie Hellman problem is hard.

Suppose there exists some adversary against ElGamal encryption. Given only public information, ie the domain parameters and a ciphertext, this adversary will recover the message within the cipher. We intend to exploit this adversary to solve the Computational Diffie Hellman problem, that is send it g^x and g^y, and somehow recover g^{xy}.

Let h, the public key, be equal to g^x. Choose $c_1 = g^y$ and $c_2 = 1$. This is entirely valid, since the oracle must account for whatever random number, k, the encryption process chooses. If $c_1 = g^y$, then it must be the case that $k = y$. If $c_2 = 1$, it must be the case that $mg^{xy} = 1$. Hence the oracle must return $m \cong g^{-xy} \mod p$. Finding the modular inverse of m is perhaps a little slow, but is not computationally hard, hence $m^{-1} \cong g^{xy} \mod p$, and we have easily solved the Computational Diffie Hellman problem.

This gives us a contradiction, since the Computational Diffie Hellman problem is hard, so the supposed OW-CPA adversary against ElGamal cannot exist.

QED

ElGamal is IND-CPA secure under the assumption that the Decisional Diffie Hellman problem is hard.

Suppose there exists some IND-CPA adversary against ElGamal. Given only public information, this adversary can send us a pair of messages, we can "encrypt" one under ElGamal, and it will tell us which one we have "encrypted". This quote marks are necessary, since we won't be performing standard ElGamal encryption. We will exploit this adversary by packaging up g^x, g^y and g^z in our challenge, and using the results to solve the Decisional Diffie Hellman problem for us.

Start by sending a public key of g^x to the adversary. When the adversary sends m_0 and m_1, return $c = (g^y, m_0 g^z)$. This means that the k chosen in ElGamal encryption is equal to y, and so for this ciphertext to correspond to m_0, it must also be the case that $z = xy$.

If the adversary returns $b = 0$, then this ciphertext must correspond to m_0, and so $z = xy$. If $z \neq xy$, there is no defined behavior for the adversary, but it cannot return $b = 0$ since this would be incorrect. From this information, we can then easily solve an instance of the Decisional Diffie Hellman problem.

This gives us a contradiction, since the Decisional Diffie Hellman problem is hard, so the supposed IND-CPA adversary against ElGamal cannot exist.

QED

ElGamal is not OW-CCA secure, since it is malleable. Given $c = (g^k, mh^k)$, we can compute $c' = (g^k, 2mh^k) \neq c$. Sending c' to the decryption oracle will yield $m' \cong 2m \mod p$. From here, m is trivially recoverable.

In summary, ElGamal is secure against all CPAs, and against no CCAs.

6.8 A Brief Word on Proofs

This chapter has been heavy on proofs and other associated mathematical arguments, and it is probably tempting to attempt to memorise them. This is almost certainly a bad idea. Being able to recite proofs from memory is usually worthless. I've tried to be as verbose as possible in describing these proofs, so that their details and associated techniques are clear. The exam is far more likely to test on techniques than it is to test on memorisation.

Chapter 7

RECAP

7.1 Block Ciphers

Block ciphers cut the message into smaller blocks to be encrypted. The **Electronic Code Book** encrypts them all individually. **Cipher Block Chaining** XORs each block of the message with the previous blocks ciphertext. The first message block is XORed with the *Initialisation Vector* and that is sent in plain text as the first block of the ciphertext. **Counter Mode** generates a pseudo-random bit-string to use in a *One Time Pad*. The IV is again sent in plaintext and is the seed for the cipher-stream. Each block adds one to the counter and encrypts it to get a bit-string to XOR with the message.

Scheme	Mode	Passes up to security model for	
		IND	OW
Block Cipher	Electronic Code Book	—	CPA
	Chained Block Cipher	CPA	CPA
	Counter	CPA	CPA

Chapter 8

PROBLEMS CLASSES

8.1 Problem Class 1

...

8.2 Problem Class 2

1. Flip it!

x	1	2	3	4	5	6	7	8
$\pi(x)$	2	4	6	1	8	3	5	7

   `GENTLEMANDONODRE`

2. N chained one-to-one substitutions can always be replaced with a functionally identical single substitution. Therefore no more secure.

3. Same as above

4. Sub and perm commutative so $S(P(S(P(m)))) = S(S(P(P(m))) \equiv S(P(m))$.

8.3 Problem Class 3

1. IND-CPA must be OW-CPA. The problem of OW-CPA is to decrypt a ciphertext with an encryption oracle. The problem of IND-CPA is to distinguish one ciphertexts from two messages given an encryption oracle. If we have a method to break OW-CPA then we can use this to break IND-CPA, therefore IND-CPA can be reduced to OW-CPA.

2. No

3. Every time we use the KEM we get a new key, so an encryption oracle would be meaningless, every time you call the oracle you get a new DEM key. We could have an oracle but it would be meaningless and not helpful.

4. Add stuff on end, decrypt, get rid of stuff. Because the encryption only uses the context of the previous blocks. Need MAC to make strong.

5. Probabilistic, but still only reliant on previous blocks (though in this case not the values of the blocks, just the number of blocks). Attack approach same as above; simply add a block on the end and then ask them to decrypt $m + block$.

6. Literally just send a message starting with a 1, and a message starting with a 0. The result leaks this!

7. This means we can decrypt two ciphertexts into one message; so we just flip the bit of the generated ciphertext and decrypt that.

8.4 Problem Class 4

1. CBC-MAC is like CBC but we completely discard every block but the last. That last block is what we use as the MAC. The proposed CFB mode's last block would be identical to the CBC-MAC due to the similar nature through which they work.

2. Collision resistance implies 2nd preimage resistance because intuitively 2nd preimage resistance is reducible to collision resistance; that is to say it is no harder than collision. If, an adversary can't find ANY two inputs that hash to the same output, then given an input, they aren't going to be able to find a second input that hashes to the same. However, for 1st preimage, they don't need to find a *second* input, just *one input*. While its unlikely that'll be easy, its not guaranteed to be as hard either.

3. (?? Maybe ??) $MAC(m||m') = h(MAC(m)|m')$, that is, a MAC generally operates in a block mode, ignoring all but the last blocks output. So to calculated the MAC of $m||m'$ we are in effect taking the the MAC of m and 'carrying on' with each block of m'.

8.5 Problem Class 5

1.
$$24140 = 1 * 16762 + 7378$$
$$16762 = 2 * 7378 + 2006$$
$$\dots$$
$$68 = 2 * \mathbf{34} + 0$$

2.

3.

4.

8.6 Problem Class 6

1.
2.
3.
4.
5.
6.
7.