Introduction to Cryptography: Summary Document

Sean al-Baroudi

April 21, 2022

Introduction and Data Security:

Overview and Symmetric Cryptography:

- **Def: Cryptography:** science of secret writing - with the goal of hiding the meaning of the message from others.
- **Def: Cryptanalysis:** science/practice of breaking crytographic systems.
- Cryptanalysis is necessary to ensure that our proposed crytographic algorithms are successful.
- **Branches of Cryptogrpahy:**
- ☐ **Symmetric Algorithms:** Algorithms that rely on two or more parties holding a secret key, used to encrypt and decrypt between plaintext and ciphertext.
- ☐ **Asymmetric Algorithms:** Public and Private Key methods - where only one party has the private key used to encrypt messages. This allwos for digital signatures.
- ☐ Crytographic Protocols: Internet and Communication protocols built on crypto algorithms, such as TLs or HTTPS
- **Symmetric Key Schemes: Contain the following Elements:**
- Plaintext (x), Ciphertext (y), Encryption Algorithm (e()), Decryption Algorithm (d()), secret key (k).
- Two parties (Alice and Bob), and an adversary (Oscar).
- Communicate over a *channel*, which may be eavesdropped by Oscar.
- Channel is considered insecure, and text (X) between Bob and Alice must be encrypted to a ciphertext (Y).
- Requires an additional secure channel to transmit secret key (K) from Alice to Bob.
- **Def: The Substitution Cipher:** Letters are mapped to other letters in a fixed mapping, and we swap letters to encode our ciphertext.
- **Brute Force Attack:** Exhaustive Key Search: Try every possible key in the key space K = $\{k_1, ..., k_n\}$. If the ciphertext makes sense after applying the decryption algorithm, you have found the key.
- **Letter Analysis Attack:** As languages don't use letters uniformly, we can tabulate the frequencies of characters, and relate them to the most common letters, to partially decode the ciphertext.
- In English, the most common letters are : E, T, A, O , S...We can also attack two and three letter words (in, the, on) quite easily.
- **Key Space** for English Substitution Cipher: 26! (factorial). The Letter Analysis attack brings down the key space significantly, as we can use human intuition to guess the mapping.

Cryptanalysis and Modular Arithmetic:

- For a good cryptographic algorithm, the ciphertext should still be secure, even if the attacker has the algorithm code (but not the key).
- Cryptanalysis involves various forms of attack on both the algorithm, participants and channels that are used. There are subcatagories:
- **Classical CryptAnalysis:**
- ☐ Mathematical Analysis: The use of statistics and mathematics to deduce a secret key, and break the ciphertext.
- ☐ Brute Force: Searching the key space sequentially until the correct key is found.
- ☐ In practice, MA is often used to reduce the keyspace, and BF is used when the reduced keyspace is within the scope of what is computable.
- **Implementation Attacks:** "Side channel attacks", those that exploit hardware or other metaphysical properties to gain information about the secret key. These are usually employed against hardware cryptographic systems.
- **Social Engineering:** Often times, the weakest link in a secure communication protocol is human beings. This attack spoofs priviledge/credentials to get inside participants to leak secret information.
- **Kerckhoff's Principle:** A cryptosystem should be secure even if an attacker knows all details of a system (except a secret key).
- **Security by Obscurity** is not good practice - release your algorithms to the public and allow people to find exploits.
- **Def: Modulo Operation:** Let a, r, m $\in \mathbb{Z}$, and m > 0. We will write:

$$a \equiv r mod m$$

if m divides a - r. We call m the modulus and r the remainder. By the division and remainder theorem (Number Theory), it is always possible to write (for $0 \leq r < m$):

$$a = qm + r$$

Where $a - r = qm$

- Our clock (modular division), divides numbers into Equialence classes. Our remainders simply denote the name of each class. Each class is countabily infinite.
- Note that remainders are not unique. For $\{0....m-1\}$, numbers outside this range are mapped to these denoted equivalence classes.
- Useful Rules for Modular Division:
- (A + B) mod m = (A mod m) + (B mod m)
- (AB) mod m = (A mod m)(B mod m)
- $a^b mod m = a^{\frac{b}{k_1}}$
-

Chapter 4: AES:

History and Overview:

- AES was developed in the mid-to-late 90s, to replace an aging DES which was becoming susceptible to brute-force attacks.
- 3DES was insufficient as a replacement, as its software implementation was not efficient (in particular, permutations) - and was 3x slower than DES itself.
- key-whitening was a simple idea, but suffered from the same issues.
- unlike DES, AES had an open-development forum and submission process. Many reserchers and industry experts were permitted to
- AES is a 128 bit symmetric-key block cipher, with optional 128 / 192 / 256 bit keys for encryption. Larger keys correspond to higher levels of security.
- AES 256 is still safe from Quantum Computing, for now.
- After a few rounds of researcher/corporate submissions,the Rinjadel (Joan Daemen and Vincent Rijmen) algorithm was selected for AES - on Oct 2nd 2000.
- **Differences between AES/DES:**
- ☐ Multi-key sizes for added encryption needs.
- ☐ The full state (128 bits) is encrypted in every round (not just 1/2 the input).
- ☐ Not a Feistel-network implementation.
- **Diagram:**

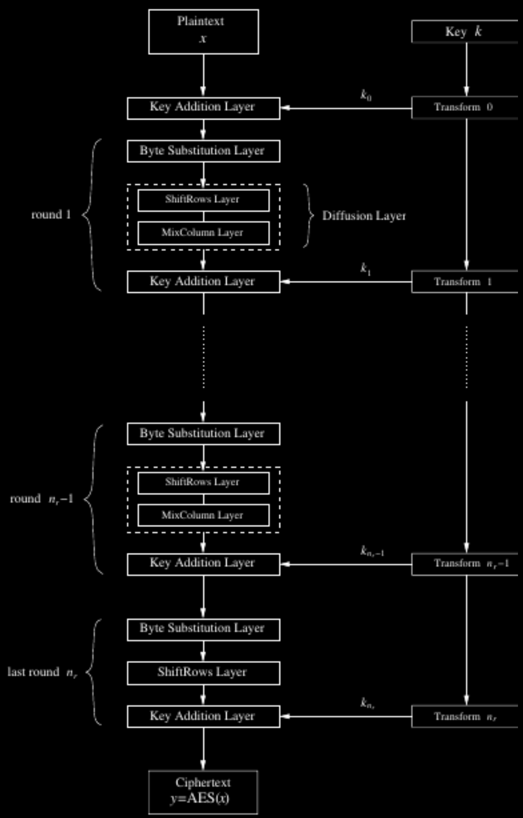


Fig. 4.2 AES encryption block diagram

- **Basic Layers of Each Round:**
- ☐ *Key Addition Layer:* Key is XORed with the state.
- ☐ *Byte Substitution Layer:* This introduces *confusion* into our ciphertext. Similar to the S-Box stuff from AES.
- ☐ **Diffusion layer:**
- ▽ *ShiftRows:* Data Permutation at the byte layer.
- ▽ MixColumn: Matrix operation that takes in 4 bytes of data, and mixes them in a defined transformation.

Galois Theory and Pre-Requisite Mathematics::

- **Def:Groups:** A set of elements G together with an operation $\circ$ which combines two elements of G. The following properties must hold:
- $\circ$ is closed for all elements of G. Specifically, $\forall$ a,b $\in$ G, a $\circ$ b $\in$ G.
- *Associativity:* a $\circ$ (b $\circ$ c) = (a $\circ$ b) $\circ$ c.
- *Identity Element:*$\exists$ 1 $\in$ G such that a $\circ$ 1 = a, $\forall$ a $\in G$
- *Inverse:* For each a in G, $\exists a^{-1}$ such that a $\circ a^{-1}$ = 1.
- **Commutativity:** a $\circ$ b = b $\circ$ a.
- **Def: Rings:**
- A ring is a field that has some of its conditions relaxed - thus is more general. The Multiplicitive operation need not be commutative, and there can be partial distributivity allowed.
-
-
- **Def: Fields:** A field $\mathcal{F}$ is a set of elements with the following properties:
- All elements of $\mathcal{F}$ form an additive group with the + operation, and have a neutral element 0.
-
- All elements of $\mathcal{F}$ form a multiplicitive group with the $\times$ operation, and have a identity element 1.
- Distributivity: Closure over elements occurs, when group operations are mixed: $\forall a, b, c \in \mathcal{F}$, a(b+xc) = ab + ac.
- In general, every field operation is a group. The ring addition operation is a group, but the multiplicitive operation is something more general (a monoid).
- Previous Theorem (Chapter 1): Proper integer fields must have a modulo divisor that is a prime number. This ensures that gcd(a,p) = 1, a $\mathcal{F}_p$ and so each element will have a proper inverse. Without this, point 2 for the definition of Fields is broken.
- Theorem 4.3.1: A field with order m only exists if m is a prime power (m = p^m) for some positive integer n and prime p. P is called the characteristic of a finite field.
- Note: A field with modulo divisor p^m will still some elements where gcd(a, p^m) = p. We must map the multiplication group to Polynomials to ensure there are proper inverses for each element in the set.
- **Theorem 4.3.2** Let p be a prime. The integer ring $\mathbb{Z}_p$ is denoted a GF(p) and is referred to as a prime field, or as a Galois Field with a prime number of elements. All non-zero elements of GF(p) have an inverse. Arithmetic in GF(p) is done modulo p.
- For regular prime fields, just do arithmetic and multiplication as you did in Chapter 1.
- Def: Extension Field GF(2^m): A Galois field in which each element is interpreted as a a polynomial with coefficents in GF(2). Note that these polynomials can be represented by bit vectors, and the maximum degree of the polynomial terms is m-1.
- Example: The number $(156)_{10}$ in GF(2_8) is $(10011100)_2$ in binary, or the polynomial:
$A(x) = x^7 + x^4 + x^3 + x^2$
- **Addition and Subtraction:** Let A(x), B(x) $\in GF(2^m)$. The sum of the two elements:

$$C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_i x^i \quad \text{where} \quad c_i \equiv a_i + b_i mod 2$$

Subtraction is defined the same way (for mod 2 - subtraction and division produce identical results).
- **EF Multiplication (Plain):** For two polynomials, we multiply as normal:

$$A(x)B(x) = (a_{m-1}x_{m-1} + ... + a_0)(b_{m-1}x_{m-1} + ... + b_0)$$

$$= C'(x) = c'_{2m-2}x^{2m-2} + ... + c'_0$$

Where the coefficients c'$_i$ are multiplied and reduced in GF(2). Note, that for a field of size 2^m, our maximum polynomial degree is m-1. We can easily jump over degree m-1 when we multiply two polynomial representations in this field. Hence, modulus over an Irreducible Polynomial (via polynomial division and Euclidian Algorithm) are required:
- Def: Extension Field Multiplication: Let A(x), B(x), $\in$ GF(2^m) and let:

$$P(x) \equiv \sum_{i=0}^{m} p_i x^i, \quad \text{for} \quad p_i \in GF(2)$$

be an irreducible polynomial. Multiplication of the two elements A, B is performed as:

$$C(x) \equiv A(x)B(x) mod P(x)$$

Where mod P(x) follows polynomial division.
- To perform the polynomial division - use the long division format done in high school. This can be coded algorithmically.
- For EF(2^m) and the AES specification, our chosen irreducible polynomial is:

$$P(x) = x^8 + x^4 + x^3 + x^1 + 1$$

- Inversion in GF(2^m): For AES, this is done by finding a polynomial reprsentation such that:

$$A^{-1}(x)A(x) = 1 mod P_I(x)$$

Practically, this is done with the lookup table that is hardcoded into the hardware/software implementations.
-

Chapter 5: More about Block Ciphers:

Practical Encryption with Block Ciphers:

- Block Ciphers are used in different ways to encrypt data. These include:
- **Block Encryption Methods:** Electronic Codebook (ECB), Cipher Block Chaining Mode (CBC)
- **Stream Encryption Methods:** Cipher Feedback Mode (CFB), Output Feedback Mode (OFB), Counter Mode (CTR)

- **Authentication of Message:** Galois Counter Mode (GSM).
- Some general points are listed and each Mode is touched upon, below:
- Note: CFB, OFB and CTR are "stream ciphers" because e() is also used in place of d() - just like we learned in Chapter 2. It is **not because** we encrypt things "one bit at a time".
- Block ciphers are a sufficent buliding block for other things incl: Hash Functions, Message Authentication Codes, and Key Establishment Protocols, Pseudo-Random Number Generators, etc.
- **Two goals:** (i) keep messages sent confidential, and (ii) know if a middle-party has tampered with our message (fraud/error detection).
- ECB and CFB require message to be exactly block sized in order to be used - padding the end of the message with unique padding signature (1000000000....) is used to accomplish this.
-

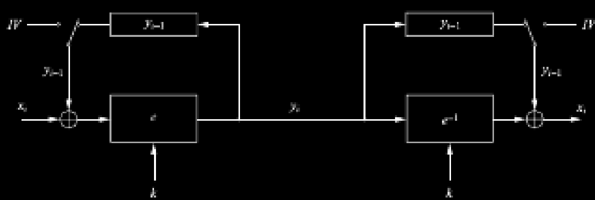
Electronic Codebook Mode (ECB):

- **Def 5.1.1:** Let e() be a block cipher of block size b, and let x_i and y_i be bit strings of length b,
  Encryption: $y_i = e_k(x_i), \ \ i \geq 1$
  Decryption $x_i = e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)), \ \ i \geq 1$

- We simply take a block of text, and encrypt it, and then send it to the recipient.
- It is called a "codebook", because it maps inputs unique outputs (1-1) deterministically.
- **Benefits:**
- + No need for block synchronization - all blocks are cryptographically $\perp$
- + Simplest of all to arrange.
- + Can easily be parralelized.
- **Drawbacks:**
- – Encyrption is deterministic. If two inputs are the same, they will map to the same output. Repeated headers/blocks will show up on the ciphertext side, and patterns can be seen.
- – Vulnerable to traffic (statistical) analysis.
- – Suceptible to Substitution atacks.
- – deterministic mapping allows for some semantic/content leakage (ex: images).
- **Block Substitution attacks:**
- Once an attacker can detect a particular mapping ($x \rightarrow y$), or detects a structre in intercepted data, they can substitute blocks and resend the message to a recipient.
- **Toy Example:** An attacker probes two banks by sending a money transfer over and doing traffic analysis. He pairs his account number with a valid ciphertext. He then intercepts an transfer message, and swaps another account number for his encrypted bank number instead. The money is now transfered to him.
- **Semantic Leakage:** because of deterministic mapping, images do not appear as noise when they are encrypted. Large areas of an image that are the same colour, will appear the same when the ciphertext is visualized.
-
- Determinism causes significant problems for this Mode. We next examine schemes that use an initialization vector (IV) and feedback to foster non-determnism.
-

Cipher Block Chaining Mode (CBC):

- with CBC, we create non-deterministic outputs by XORing the y_ith ciphertext with the x_{i+1} plaintext. This severely restricts traffic analysis.



-
- **Definition 5.1.2:** Cipher Block Chaining Mode: Let e() be a block cipher of block size b, and let x_i and y_i be bit strings of length b, and IV be a nonce of length b:
  Encryption (1st Block): $y_1 = e_k(x_1 \oplus IV)$
  Encryption (General): $y_i = e_k(x_i \oplus y_{i-1}) \ \ i \geq 2$
  Decryption (1st Block): $x_1 = e_k^{-1}(y_1) \oplus IV$
  Decryption (General): $x_i = e_k^{-1}(y_i) \oplus y_{i-1}$
- Mode Verification (General Case):

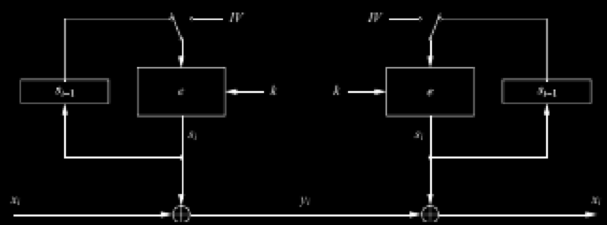
$$d(y_i) = e_k^{-1}(y_i) \oplus y_{i-1} = ...$$

$$e_k^{-1}(e_k(x_i \oplus y_{i-1})) \oplus y_{i-1} = x_i \oplus y_{i-1} \oplus y_{i-1} = x_i$$

- We must change the IV fairly frequently. As long as a number of different ciphertexts don't use the same IV, traffic analysis becomes difficult to perform.
- In most cases, it is safest to make the IV a nonce. We can transmit the IV as a plaintext, as the attacker does not have the key K - making things simple.
- We could also use a counter or seed+pseudorandom counter to genreate our IV's.
- Is CBC resistant to Substitution attacks? If IV is not changed enough, no - in a session an attacker might detect a pattern, and try to inject bad blocks into the message stream, much like we saw with ECB.
-

Output Feedback Mode (OFB):

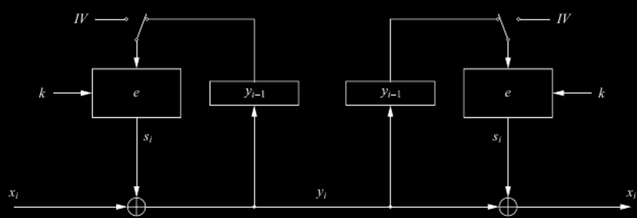
- Here, our XORed input is generated via the encryption algorithm, and the plaintext is XORed with it. The encryption algorithm doesn't directly touch the ciphertext/plaintext.



-
- **Definition 5.1.2:** Cipher Block Chaining Mode: Let e() be a block cipher of block size b, and let x_i and y_i be bit strings of length b, and IV be a nonce of length b:
  Encryption (1st Block): $s_1 = e_k(IV) \ \ , y_1 = s_1 \oplus x_1$
  Encryption (General): $s_i = e_k(s_{i-1}) \ \ , y_i = s_i \oplus x_i \ \ i \geq 2$
  Decryption (1st Block): $s_1 = e_k(IV) \ \ , x_1 = s_1 \oplus y_1$
  Decryption (General): $s_i = e_k(s_{i-1}) \ \ , x_i = s_i \oplus y_i \ \ i \geq 2$
- This is a stream cipher, but still generated in a block-wise fashion
- **Advantages:**
- ++ Block Cipher seeds are $\perp$ to plaintext. Once can precompute many si's to help speed up the e()/d() process.
- As usual, IVs should be nonces (if possible).
- **Verification (of decryption):**
-
-

Cipher(text) Feedback Mode (CFB):

- Similar to OFB, however the ciphertext is fedback, instead of a seed si.

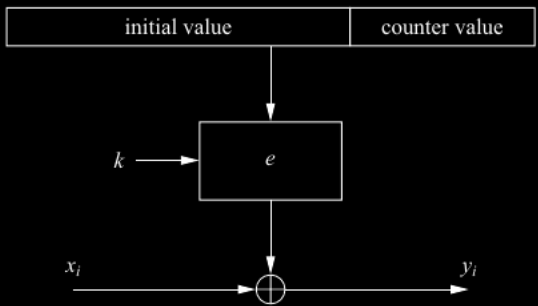


- **Definition 5.1.4:** Cipher Block Chaining Mode: Let e() be a block cipher of block size b, and let x_i and y_i be bit strings of length b, and IV be a nonce of length b:
  Encryption (1st Block): $y_1 = e_k(IV) \oplus x_1$
  Encryption (General): $y_i = e_k(y_{i-1}) \oplus x_i \ \ i \geq 2$
  Decryption (1st Block): $x_1 = e_k(IV) \oplus y_1$
  Decryption (General): $x_i = e_k(x_{i-1}) \oplus y_i \ \ i \geq 2$
- **Advantages:**

- ++Again. we can precompute the si seeds that we compute with e().
- ++Fairly simple to implement.
- **Disadvantages:**
- – Cannot be parallelized (non /*perp*) inputs! Our blocks have an ordering for processing.
-

Counter Mode (CTR):

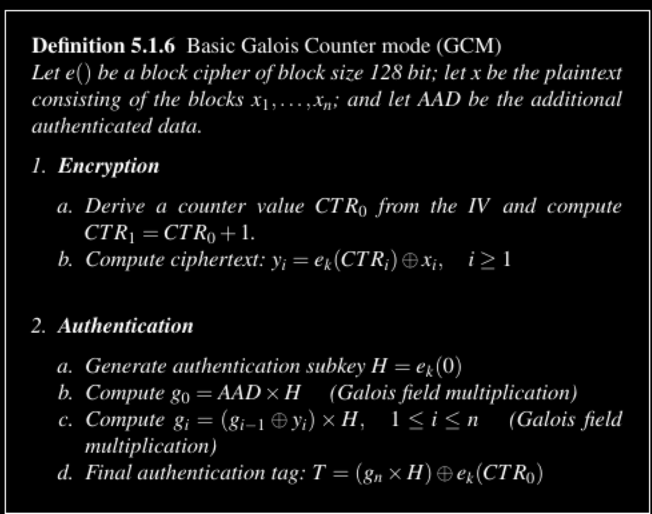
- This is also a simple implementation, with usual IV and an appended counter that is placed in a 128-256 wide bit window (for AES e() function).



-
- **Definition 5.1.5:** Let e() be a block cipher of block size b, and let x_i and y_i be bit strings of length b. The concatenation of the IV and the counter CTR$_i$ is denoted by (IV || CTR$_i$) and is a bit string of length b
  Encryption: $y_i = e_k(IV||CTR_i) \oplus y_i \quad i \geq 1$ Encryption: $x_i = e_k(IV||CTR_i) \oplus y_i \quad i \geq 1$
- With counter mode, we can share the (IV || CTR$_i$) publically, as our attacker (Oskar) does not have access to the key of Alice and Bob.
- An advantage of this mode is that it is very parallelizible.
- A disadvantage is that there are limits on the amount of data we can encrypt - this is dependent on when our counter saturates and rolls over. In general, we can encrypt 8×2^p bytes of data, where p is the bit window length for the counter. For very large pieces of data, this may not be ideal. You could just transmit multiple IVs - to overcome this.
-

Galois Counter Mode (GCM):

- I don't pretend to fully understand this - a screenshot of all relevent details is below.

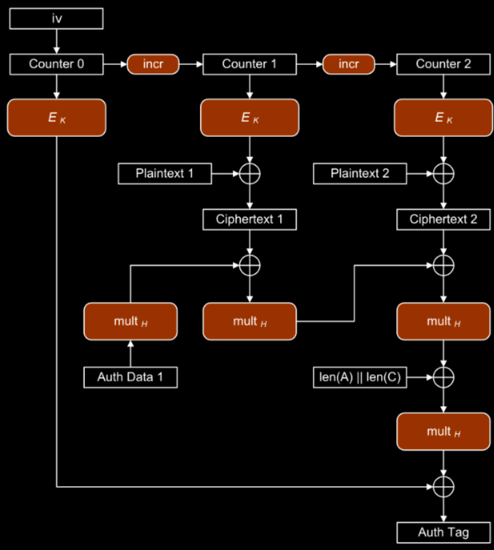


Definition 5.1.6 Basic Galois Counter mode (GCM)
Let e() be a block cipher of block size 128 bit; let x be the plaintext consisting of the blocks $x_1, \ldots, x_n$; and let AAD be the additional authenticated data.

1. ***Encryption***
   a. *Derive a counter value CTR_0 from the IV and compute $CTR_1 = CTR_0 + 1$.*
   b. *Compute ciphertext: $y_i = e_k(CTR_i) \oplus x_i, \quad i \geq 1$*

2. ***Authentication***
   a. *Generate authentication subkey $H = e_k(0)$*
   b. *Compute $g_0 = AAD \times H$ (Galois field multiplication)*
   c. *Compute $g_i = (g_{i-1} \oplus y_i) \times H, \quad 1 \leq i \leq n$ (Galois field multiplication)*
   d. *Final authentication tag: $T = (g_n \times H) \oplus e_k(CTR_0)$*

-



-
- **Advantages:**
- ++ Parallelizable
- ++ Includes MAC Checking
- ++ Checks all the boxes that cryptographers like
- **Disadvantages:**
- –Hard to implement, and fully understand
- Note: the A/D claims above come from cryptographer blogs talking about GCM. It is the most desirable choice out of all modes, but nobody enjoys implementing it or checking its correctness (!!).
-

Key Search Revisted / Increasing Security:

Brute Force Key Searching, part II:

- It is not always true that, because we have 2^k keys, that we need to brute force all of them.
- We can have multiple, or false-positive keys that match a plaintext to a cipher text. This can occur if:
- We have a (p/c) pair that is a particularly short length
- If the number of (p/c) pairs is smaller than the number of keys.
- **Theorem 5.2.1:** Given a blck cipher with a key length of k bits and block size of n bits, as well as t (p/c) pairs, the expected number of false keys which encrypt all plaintexts to a corresponding ciphertexts is:

$$2^{k-tn}$$

- **Note:** This assumes that an encryption algorithm $e_k()$ maps a plaintext to a ciphertext in a random and uniform fashion (we don't get bunching, or miss parts of the range.
- Example: Suppose we have a 80 bit key, and 64 bit block size. The average number of key matches for a given (p/c) is: $2^{80-64} = 2^16$. If, we have two (p/c) pairs, then the "likelihood" of getting a double false positive is $2^{80-128} = 2^{-48}$.
  So we can be sure that if we have a key that decodes both (p/c) pairs, that it is the correct one.
-
-
-
-

Failure of 2x Encryption, MITH Attacks:

- Surprisingly, double (but not 3x...nx) encryption is not effective in increasing our keyspace, and message security.
- 2x Encryption vulnerable to Meet-in-the-Middle Attacks
- **Example:** Consider a DES double encryption scheme. We have two keys K_L and K_R, and perform the following Encryption:

$$y_i = e_{K_R}(e_{K_L}(x_i))$$

  We will show that this is vulnerable to MitM attack.
- Theorem 5.3.1: This is a modified version of Theorem 5.2.1, for when we have multiple keys. Our statement - on the expected number of false keys - changes to:

$$2^{lk-tn}$$

Where l is the number of keys.

- Suppose we have a 2x encryption. An attacker can perform the following:
  1. This attack assumes the attacker has a few (p/c) pairs only - to rule out false positives.
  2. Oscar first compiles a table intermediate results using the first (Left) key. He takes a plaintext and encrypts each one with k$_L$, to yield intermediate results z_{Lj}:

     $$\{(z_1, k_{L1}), ...(z_n, k_{Ln})\} \text{ for } n = 2^k$$

  3. Once the intermediate table of results are encrypted, Oscar, then decrypts the first ciphertext, iterating through all possible keys of k$_R$.

     $$z'_{Lj} = d_{K_{Rj}}(x_1)...$$

     He compares the intermediate output to the table of results he saved previously.
  4. Once a match (or "collision") is discovered, he then takes the other pairs of (p/c)'s and tries to match intermediate results, to rule out a false positive.
  5. If he does not get a match on the other (p/c)'s, he starts his decryption search again. Else, **he has found the key.**
- From this attack, Oscar at most has to compute $2^k + 2^k = 2^{k+1}$ keys for each side of the attack. Our key length therefore is not 2^{2k} as we might have hoped!
- This attack in principle is more difficult, because Oscar has to maintain O((kx)2^k) entries in a datastructure for his search. If k is big enough, this requires a lot of storage and overhead.
-
-
-

3x Encyrption Saves the Day:

- In practise, this is implemented as:

  $$y = e_{k_1}(e^{-1}_{k_2}(e_{k_3}(x)))$$

  If we let k1 = k2, we have the option to perform single encryption instead. This is known as Encryption-Decryption-Encryhption (EDE).
- Triple Encryption works, because for Oscar to Meet in the Middle, he has to perform a double encryption in one of the directions. This means he has to try $2^n + 2^n * 2^n$ keys instead of just 2^{n+1}.
- In general, Triple Encryption with DES effectively doubles the key length to 112 bits - making it much harder to attack.
- In general, when n-ary encryption is done, MITM attack can only take down the number of keys to try by a factor of 2^n. With just 3 or more layers of encryption, MitM attack is rendered useless!

Key Whitening:

- Is a computationally simple way of enlarging our key space. Diagram with equations below:

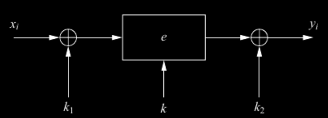


Fig. 5.12 Key whitening of a block cipher

Definition 5.3.1 Key whitening for block ciphers
Encryption: $y = e_{k,k_1,k_2}(x) = e_k(x \oplus k_1) \oplus k_2$
Decryption: $x = e^{-1}_{k,k_1,k_2}(x) = e_k^{-1}(y \oplus k_2) \oplus k_1$

- If our XOR key lengths are n bits, our effective key search space is of size:

  $$2^{K+2n}$$

  For a raw brute force attack by an attacker. If a MitM attack is performed, this can be reduced to:

  $$2^{K+n}$$

  keys to test for Oscar.

Introduciton to Public-Key Cryptography:

VS Symmetric, and Basic Concepts:

- **Problems with Symmetric Cryptography:**
- ☐ Securing a channel initial key distribution.
- **Key Management:** Each pair of parcitipants needs their own key to communicate. Network effects. N participants need $\frac{n(n-1)}{2}$ keys (quadratic growth of keys).
- No Protection against Cheating (Non-Repudiation): Both A and B have the same abilities. Who is to say that B can't send a payment to C, and then claim that (i) it is A who is going to make the payment, or (ii) that B never made a request to pay (forgery)?
-
- **Def (Legal): Non-Repudiation:** "refers to a situation where an author cannot successfully dispute the authorship or validity of a contract. To repudiate is to reject or deny the validity of. **We want non-repudiation** to prevent someone from disowning a contract or the validity of a claim.
- **Asymmetric Key Concept:** Let B have a private and public key. Let all participants have the public key. Then each participant can send a message to B using the public key - only Bob can decrypt the message using his private key (asymmetry of abilities).
- **Analogy:** Locked Mail Box - only Bob has the key.
- Using the AKC, we have a basic method for creating a secure channel, for Symmetric Cryptography. Alice can encrypt her key k$_S$ using the asymmetric public key, and only Bob can decrypt it.
- **Def: 6.1.1: One-way Functions:** A function f() is one-way if:
  (1) y = f(x) is computationally easy.
  (2) x = f$^{-1}(x)$ is computationally hard
  Where "computationally easy" means calculable in poly-time, and infeasible is not in poly-time.
-

0.1 Security Mechanisms:

- There are four main functions that Asymmetric Cryptoraphy can provide:
  (1) Key Establishment: Creating a secure channel to distribute secret keys.

  (2) Non-Repudiation: Prevent participants from repudiating - ensure message integrity via digital signatures.

  (3) Identification: ID participants with challenge-and-response + Digital Signatures.

  (4) Encryption of Messages.

- Note that Symmetric Cryptography **has trouble** with (1) to (3).
- Performance Comparison: Symmetric Algorithms are about 100-1000x faster than Asymmetric Algorithms.
- **Hybrid strategy (in practice):** Use Asym. for functions (1 - 3). Use Symm. for (4), the encryption of messages.
- **Important Public-Key Algorithms:**
- ☐ RSA: Integer-Factorization (of large primes)
- ☐ Diffie-Hellmen Key Exchange: Discrete Logarithm Problem.
- ☐ Elliptic Curve Digital Signature Algorithm (ECDSA): Uses Discrete Logarithm on Elliptic Curve spaces (generalization).
- **Complexity of Asymmetric Algorithms:** Let b the the number of bits in length of a given asym. key. Then the runtime is of order $O(b^3)$.
-
-
-

Essential Number Theory:

- Def: gcd(r$_0$,r$_1$): Let r$_0$ and r$_1$ be natural numbers. These numbers can be written as a unique product of primes, by the Fundamental Theorem of Arithmetic. **GCD is the product of all common prime factors**
- Key Result (for lemmas): Given gcd(r$_0$, r$_1$), we may write r$_0$ = gx, and r$_1$ = gy, where "g" is our common prime factors, and x and y separate factors that do not overlap.

- **Result 1:** gcd(r_0,r_1) = gcd(r_0 - r_1,r_1), for $r_0 > r_1$.

  This is because gcd(r_0 - r_1,r_1) = gcd(g(x - y), gy) = g
- We can iteratively reapply Result 1 to show that gcd(r_0,r_1) = ... = gcd(r_0 - mr_1,r_1), where r_0 - mr_1 > 0.
- **Result 3:** gcd(r_0,r_1) = gcd(r_0 mod r_1,r_1).
  This is an extention of Result 2: if we set our "m" to be the maximum value possible, we end up with the definition of modular division - with our remainder denoting an equivilence class.

- **Result 4: The Euclidian Algorithm Idea:** gcd(r_0,r_1) = gcd(r_1, r_1 mod r_0) = ... gcd(r_l,0) = r_l.

  The idea is that we can reduce the problem of finging the GCD of two larger numbers (r1 and r0) to smaller numbers with suitable modular division. Intuitively, as we iterate with further modular divisions, we "cut away" the non "g" common factors, until all we are left with is g.
- **Standard Euclidian Algorithm:**

  Euclidean Algorithm
  Input: positive integers r_0 and r_1 with $r_0 > r_1$
  Output: $\gcd(r_0, r_1)$
  Initialization: $i = 1$
  Algorithm:

  1 DO
  1.1 $i = i + 1$
  1.2 $r_i = r_{i-2} \bmod r_{i-1}$
  WHILE $r_i \neq 0$
  2 RETURN
  $\gcd(r_0, r_1) = r_{i-1}$

- **Extended Euclidian Algorithm:**

  Extended Euclidean Algorithm (EEA)
  Input: positive integers r_0 and r_1 with $r_0 > r_1$
  Output: $\gcd(r_0, r_1)$, as well as s and t such that $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$.
  Initialization:
  $s_0 = 1 \quad t_0 = 0$
  $s_1 = 0 \quad t_1 = 1$
  $i = 1$
  Algorithm:

  1 DO
  1.1 $i = i + 1$
  1.2 $r_i = r_{i-2} \bmod r_{i-1}$
  1.3 $q_{i-1} = (r_{i-2} - r_i)/r_{i-1}$
  1.4 $s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}$
  1.5 $t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$
  WHILE $r_i \neq 0$
  2 RETURN
  $\gcd(r_0, r_1) = r_{i-1}$
  $s = s_{i-1}$
  $t = t_{i-1}$
- **Def 6.3.1: Euler's Phi Function:** A function that gives the number of integers in $\mathbb{Z}_m$ that are relatively prime to m is denoted by $\Phi(m)$
- **Theorem 6.3.1:** LEt m have the following canonical factorization, as per the FTA:

$$m = p_1^{e_1} ... p_n^{e_n}$$

  where p's are distinct primes, and exponents e are positive numbers. Then Euler's Phi:

$$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$$

- Euler's Phi is useful for calculating how many relatively prime (read: inversable) elements there are in $\mathbb{Z}_p$.
- **Theorem 6.3.2: Fermat's Little Theorem:** Let a be an integer, and p be a prime number. Then:

$$a^p; \text{ (mod p)} \equiv \quad a; \text{ (mod p)}$$

- More specifically, we can use this to calculate a^p very quickly. We can also find ring inverses, via the modified formula:
- Note that FLT cannot be used if our **power and ring base are not prime.**

$$a^{-1}; \text{ (mod p)} = a^{p-2}; \text{ (mod p)}$$

- **Recall:** abc mod p = (a mod p)(b mod p)(c mod p). So we can calculate things by hand very quickly.
- **Theorem 6.3.3: Euler's Theorem:** Let a and m be integers with gcd(a,m) = 1. Then:

$$p^{\Phi(m)} \equiv 1 (mod m)$$

  Where the exponent is the Euler Phi function.
- In addition to the ET, we can modify it to find inverses:

$$a^{\Phi(m)-1} \equiv a^{-1} (mod m)$$

- In practice, the Extended Euclidian Algorithm is usually used to quickly calculate inverses. These methods above are used in corner cases.
-

RSA Cryptosystem:

Enc and Decr, Key Gen:

- RSA was the first available Asymmetric Encryption Scheme.
- **Main Usage:**
- + Encrypting small pieces of data (key transport).
- + Digital Signatures, Certificates and ensuring non-repudiation.
- **Drawbacks:**
- - Several Times slower than Symmetric (cube of bit length O(n^3) time to encrypt).
- - Choosing key set can be computationally intensive.
- - Parameters (keys, exponents) are typically quite large (1024bits+).
- Underlying One-Way Problem: Integer Factorization using very large primes.
- **Ring Space for Computations:** We do modular and exponent calcualtions in a $\mathbb{Z}_n$ ring, with a plaintext x and ciphertext y. x,y < n, so we can have 1-1 mappings for decryption.
- **Encryption:** Given a public key (n,e) = k_{pub} and plaintext x, our encryption is:

$$y = e_{k_{pub}}(x) \equiv x^e mod n$$

  for x,y $\in \mathbb{Z}_n$
- **Decryption:** Given a private key (d) = k_{priv} and ciphertextt y, our decryption is:

$$x = d_{k_{priv}}(y) \equiv y^d mod n$$

  for x,y $\in \mathbb{Z}_n$
- **Key Tuples:** Our Public Key is (n,e), our Private key is (p,q,d).
- In practice, our computed algorithm parameters are (n,e) for the public key, and (d) for the private key.
- **Parlance:** e is known as a public or encyrption exponent, and d is known as a private or decryption exponent.
- **Implied Requirements for RSA:**
- (i) Attacker has access to (n,e), computation of (d) from these parameters must be computationally infeasible.
- (ii) We cannot encrypt more than $|n|$ bits, where n is the ring modulus. This means messages we send are quite short!
- (iii) As we need to do modular exponentiation for d() and e(), we need to be able to do this quickly.
- (iv) For a given n, there needs to be many (pub/priv) pairs - to avoid a brute-force attack.

Proof of Correctness:

-
-
-
-
-
-
-
-

Speedup Techniques for RSA:

Fast Exponentiation:

- Numbers are large - we need tricks to make exponentiation computationally feasible. We are dealing with a 1kb number being raised to a 1kb power, which means calculating in a human manner (x.x.x.x...) is quite slow.
- RSA uses the Square and Multiply Algorithm to accomplish this.
- Idea: To calculate x^N, we use combinations of squaring and multiplying to build up-to x^N. Specifically:

$$SQ : (x^k)^2 = x^{2k}$$
$$MULT : (x^k)x = x^{k+1}$$

- **Algorithm: Descriptive:** Take the exponent and represent it as a binary number. Scan the number from MSB to LSB (Left to Right). Use the MSB to set the first bit (always 1). Square, and look at the next bit. If the next bit is a 1, also apply a mult. Repeat the mandatory squaring, and toggled multiplication for every bit scanned until the bit string is exhausted.
- Idea: Squaring does a bit shift, and multiplying sets a bit. We are effectively reconstructing our bit string by applying these operations, when we interpret the bit string. $x^{101211\dots}$ = big number by the end of it.
- **Algorithm:**

Square-and-Multiply for Modular Exponentiation
Input:
base element x
exponent $H = \sum_{i=0}^{t} h_i 2^i$ with $h_i \in 0,1$ and $h_t = 1$
and modulus n
Output: $x^H \bmod n$
Initialization: $r = x$
Algorithm:

```
1     FOR i = t − 1 DOWNTO 0
1.1       r = r^2 mod n
          IF h_i = 1
1.2           r = r · x mod n
2     RETURN (r)
```

-
- Average Number of calculations for t bit exponent:

$$SQ + MUL = t + 0.5t = 1.5t$$

This is on average, based on a Hamming weight of 0.5 (roughly half the bits in the string are ON, on average).

-
-

Fast Encryption with Short Public Exponents (e):

- Both encryption and decryption involve exponentiation and modular division. We can reduce our computational time by upto 1/4 (for an E-D round) - by choosing short public exponents.

Public key e	e as binary string	#MUL + #SQ
3	11_2	2
17	10001_2	5
$2^{16}+1$	$1\,0000\,0000\,0000\,0001_2$	17

- Note that (e,n) make up the public key - so it doesn't matter if an attacker knows them.
- $\perp$: we can speed up our encryption without compromising our decryption (d).
- **Claim:** We cannot shorten our private key - it must be of length 0.3t at minimum, where t = $|n|$. See Question 7.8 Answer (accomanying PDF), for the answer to this.
- Note: These exponents are short, however the accompanying private exponent is NOT.

Fast(er) Decryption with Chinese Remainder Theorem:

- **Chinese Remainder Theorem:** Let N = $p_1 p_2 \dots p_n$, and each of the p's be pairwise co-prime with one another. For a set of a_i's such that $0 \leq a_i \leq N$, there are a **unique** set of x's, such that applying the Euclidian Algorithm to x_i (with divisor p_i) is mapped to the equivilence class a_i:

$$x_1 \equiv a_1 mod p_1$$

$$\dots$$

$$x_n \equiv a_n mod p_n$$

- Application. Let the ai's be our plaintext (x), and our prime factors be p and q. We find a reduced plaintext and work with this.
- **Fast Encryption/Decryption has three stages:**
- Note that because our encryption / decryption uses the same mod equaitons, we can use the below method to speed up calculations for both, in addition to using SQ/MULT for large exponentiations. The below method is written in terms of decryption, due to the titling of this subsection:
  1. **Transform into CRT Range:** We reduce the plaintext x via modulo division with our primes:

     $$y_p \equiv y mod p$$
     $$y_q \equiv y mod q$$

  2. **Exponentiation in Range:**

     $$x_p = y_p^{d_p} mod p$$
     $$x_q = y_q^{d_q} mod q$$

     with our d exponents given by:

     $$d_p \equiv d mod (p-1)$$
     $$d_q \equiv d mod (q-1)$$

     Exponents also modularly divided are bounded above by p and q, respectively, as are the two ciphertexts y.
  3. **Inverse Transform back to Domain:** Finally we must assemble y from y_p and y_q. This is done with the following calculation:

     $$x \equiv [qc_p]x_p + [pc_q]x_q$$

     Note that the terms are mixed. We can pre-compute the c's and store them:

     $$c_p \equiv q^{-1} mod p$$

     $$c_q \equiv p^{-1} mod q$$

- Computational Complexity: for SQ/MULT, 1.5t again on average. However, our primes are of length t/2, so our multiplication complexity is of order $O(\frac{t}{4})$ as it decreases quadratically with bit length.
- So practically, using the CRT speeds up our computation by about 4x.

Speed up Summary: We use Fast Exponentiation, a short public exponent, and the CRT for both encryption and decryption, to get our algorithms to run in reasonble time.

Finding Large Primes:

- Our algorithms above all depend on choosing pq to be primes. They need to be large, as our $K_{private}$ = (p,q,d).
- Our primes should be of length t/2.
- We need to use a non-predictable Random Number Generator. See Chapter 2 for more detalis...
- Some main questions to consider:
  1. How many random integers must be tested to find primes?
  2. Can we have tests / heuristics to quickly narrow it down (YES)
  3. The density of primes goes down as we increase the size of numbers. For larger keys (n=2048,4096), will there be enough primes lying around for us to find, and to make the search space too large for an attacker?
- Prime Number Theorem (Result): For a given odd number:

$$P(\text{P is Prime}) = \frac{2}{ln(p)}$$

- Primality tests **do not give** deterministic answers, they suggest one of two: (i) P is likely prime, or (ii) P is composite.
- We can increase the probability that P is prime, by performing the test with differing parameters a set number of times. Practically, we are looking for $< 2^{-80}$ that we have a false positive.
- **Fermats Primality Test:**

  Fermat Primality Test
  Input: prime candidate $\tilde{p}$ and security parameter s
  Output: statement "$\tilde{p}$ is composite" or "$\tilde{p}$ is likely prime"
  Algorithm:

  1 FOR $i = 1$ TO s
  1.1 choose random $a \in \{2, 3, \ldots, \tilde{p} - 2\}$
  1.2 IF $a^{\tilde{p}-1} \not\equiv 1$
  1.3 RETURN ("$\tilde{p}$ is composite")
  2 RETURN ("$\tilde{p}$ is likely prime")

- This Test derives from Fermats Little Theorem.
- **It is sufficient** - but not necessary - for primes to satisfy this theorem. There are composite numbers that can give similar resluts.
- This algorithm is run with mutiple values of a, to try to rule out pathological composites.
- Pathological Composites - the Carmichael Numbers (C): such that:

$$gcd(a, C) = 1$$

  and

$$a^{C-1} = 1 mod C$$

  for all integers a!
- Note that for large Carmichael Numbers, there are very few a's where FT fails (and composites are detected). As we are looking for large primes, and there are 100k CN's in [1,10^15], this causes a latent worry.
- To avoid this problem, another test is used:
- **Miller-Rabin Primality Test:**

  Miller–Rabin Primality Test
  Input: prime candidate $\tilde{p}$ with $\tilde{p} - 1 = 2^u r$ and security parameter s
  Output: statement "$\tilde{p}$ is composite" or "$\tilde{p}$ is likely prime"
  Algorithm:

  1 FOR $i = 1$ TO s
  choose random $a \in \{2, 3, \ldots, \tilde{p} - 2\}$
  1.2 $z \equiv a^r \bmod \tilde{p}$
  1.3 IF $z \not\equiv 1$ and $z \not\equiv \tilde{p} - 1$
  1.4 FOR $j = 1$ TO $u - 1$
  $z \equiv z^2 \bmod \tilde{p}$
  IF $z = 1$
  RETURN ("$\tilde{p}$ is composite")
  1.5 IF $z \neq \tilde{p} - 1$
  RETURN ("$\tilde{p}$ is composite")
  2 RETURN ("$\tilde{p}$ is likely prime")

- **Miller-Rabin Theorem:**
  Theorem 7.6.1 *Given the decomposition of an odd prime candidate* $\tilde{p}$

$$\tilde{p} - 1 = 2^u r$$

  where r is odd. If we can find an integer a such that

$$a^r \not\equiv 1 \bmod \tilde{p} \quad \textit{and} \quad a^{r2^j} \not\equiv \tilde{p} - 1 \bmod \tilde{p}$$

  for all $j = \{0, 1, \ldots, u-1\}$*, then* $\tilde{p}$ *is composite. Otherwise, it is probably a prime.*

0.2 Padding, Attacks and Implementation:

Padding:

Attacks on RSA:

- Weaknesses of RSA:
  1. Encryption is Deterministic, so traffic analysis can be performed (if repeatedly used to communicate - unlikely).
  2. If x=0,1,-1, ciphertexts are 0,1,-1. Because there is a 1-1 mapping, we know the plaintext immediately.
  3. **Malleability:** We can intercept ciphertexts and corrupt them. (Example: Corrupt a monetary amount).
- **Protocol Attacks:** Attacking poorly padded RSA, or exploiting malleability.
- **Mathematical Attacks:**
- The attacker's challenge is as follows: Knowing (n,e): Calculate the following in order

$$\Phi(n) = (p-1)(q-1)$$

$$d^{-1} \equiv e mod \Phi(n)$$

$$x \equiv y^d mod n$$

- The attacker is held at bay, because **they do not know p and q.**

Implementation Considerations:

References

[1] https://ethereum.stackexchange.com/questions/109847/how-to-install-ganache-ui-on-ubuntu-20-04-lts

[2]

[3]

[4]

[5]

[6]