Public Key Cryptography

Encryption Techniques

CS3235 - Prateek Saxena

In public-key cryptography, Alice and Bob do not share a pre-established secret. Instead, they agree on a set of parameters that are disclosed publicly, and each publish their own "public" keys which can be used to communicate with them. Alice and Bob have their corresponding "private" keys respectively, which are not the same as in symmetric-key cryptography. This is why public-key cryptosystems are also referred to as *asymmetric* cryptography. As with all modern cryptosystems, the setup and encryption / decryption algorithms are publicly known. In this lecture, we will study how to perform encryption in the asymmetric cryptographic setting.

1 Mathematical Preliminaries

We provide a very brief summary of the preliminaries. Readers are encouraged to be familiar with the facts from this cheat sheet[1], or any other sources listed on the course webpage.

Modular Arithmetic. Modular arithmetic is system of arithmetic over integers in which results "wrap around" after reaching an integer upper limit, known as a *modulus*. We will use $(+)$ to denote integer addition, $(\cdot)$ to denote integer multiplication, and the application of on an integer x repeated y times is the integer exponentiation operation, denoted as x^y . In modular arithmetic, the reduction $(mod\ N)$ operation (for a given N) on x is the remainder left after dividing x by N. It is easy to see that many integers (e.g. r, $r + \cdot N$, $r + \cdot 2N, \ldots$) all have the same remainder after dividing by N; therefore, we say that $x \equiv r (mod\, N)$ if x has a remainder r when divided by N, suggesting that x is an element of a congruence class defined by (r, N). Congruence here means that if $a_1 \equiv b_1 (mod\, N)$ and $a_2 \equiv b_2 (mod\, N)$, then $(a_1 + a_2) \equiv (b_1 + b_2)(mod\, N)$ and $(a_1 \cdot a_2) \equiv (b_1 \cdot b_2)(mod\, N)$. This congruence relation defines an equivalence relation, and related elements form a congruence class.

Modulus is distributive over $(\cdot)$ and $(+)$ in the second component — that is, $(a + b) mod\ N = ((a\ mod\ N) + (b\ mod\ N))\ (mod\ N)$, and similarly for multiplication.

Groups. Groups are way to describe algebraic structures in mathematics. A group is defined by a set $\mathcal{G}$ and a binary operation op, which satisfies the following four properties:

1. *Closure.* The elements of $\mathcal{G}$ are closed under the operation op. That is, $\forall x, y \in \mathcal{G}$, $x\ \mathsf{op}\ y$ is an element in $\mathcal{G}$.

2. *Asssociativity.* The order of evaluation of the operator does not change the result. That is, $\forall x, y, z \in \mathcal{G}$, $(x\ \mathsf{op}\ (y\ \mathsf{op}\ z)) = ((x\ \mathsf{op}\ y)\ \mathsf{op}\ z)$

3. *Identity Element.* There exists an element i which $\forall a \in \mathcal{G}$ yields $a\ \mathsf{op}\ i = i\ \mathsf{op}\ a = a$.

4. *Invertibility.* For all elements $a \in \mathcal{G}$, there exists an inverse element $a^{-1} \in \mathcal{G}$, such that $a\,\mathsf{op}\,a^{-1} = i$, where i is the identity element.

[1] https://crypto.stanford.edu/ dabo/cs255/handouts/numth1.pdf

Let us consider a few examples and non-examples of groups.

- The set of integers $\mathbb{Z}$ with $+$ as the group operation is called the additive group. The identity element is 0, the inverse of 4 is -4.
- The set of non-negative integers $\{0, 1, ..., N-1\}$ for any integer N with $\cdot$ followed by a *mod* N as the group operator is *not* a group. There is no inverse for the element 0.
- The set of positive integers smaller than and co-prime [2] to an integer N, with $(\cdot)$ followed by a reduction *mod* N as the group operator, defines a group. The group is often called the multiplicative group modulo N, and is denoted as $\mathbb{Z}_N^*$. It doesn't contain 0 which is not invertible. The identity element is 1. No elements (except 1) in $\mathbb{Z}_N^*$ share a factor with N; therefore, the multiplication of two group elements always produces numbers co-prime to N.
- The set of positive integers $\{1, \ldots (p-1)\}$ for a prime p, with $(\cdot)$ followed by a reduction *mod* p as the group operator, defines a group. The group is called a *the multiplicative group mod p*, and is written as $\mathbb{Z}_p^*$.

The *order* of a group is its cardinatility, the number of elements in the set. A group is called *cyclic* if there is an element g which under repeated application of the group operator generates all the elements of the group. Such an element g is called a *generator* or a *primitive root*. As an example, $\mathbb{Z}_p^*$ is a cyclic group. For $\mathbb{Z}_7^*$, the element 3 is a generator because $\{3(mod\, 7), 3^2(mod\, 7), \ldots, 3^6(mod\, 7)\}$ is the set $\{3, 2, 6, 4, 5, 1\}$ which is base set of the group $\mathbb{Z}_7^*$. Thus, 3 generates all elements in $\mathbb{Z}_7^*$. Not all elements in $\mathbb{Z}_7^*$ are generators — for instance, the integer 2 produces a strict sub-group $\{1, 2, 4\}$ by repeated multiplications mod 7; hence, 2 is not a generator.

Fermat-Euler Theorem. Let us look at an interesting property of such cyclic groups as $\mathbb{Z}_N^*$. We ask: what is the order of the group $\mathbb{Z}_N^*$? Specifically, given a generator g of $\mathbb{Z}_N^*$, for what value of a will $g^a \equiv 1(mod\, N)$? The answer to this puzzle is the quantity $\phi(N)$, where $\phi(\cdot)$ is called the Euler's totient function. $\phi(N)$ for an integer N is the number of positive integers smaller than N and co-prime to it. This result is a famous theorem in group theory and modular arithmetic called the *Fermat-Euler theorem*. The Fermat-Euler theorem is a classical result in modular arithmetic, and more general than the statement above. It states that given any pair of integers x and N, if they are co-prime, then

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

For a prime p, $\phi(p) = (p-1)$. The number of positive integers smaller than p and co-prime to it is all the integers $\{1, \ldots, p-1\}$. Therefore, for a primes p and a positive integer $a < p$, since a and p are co-prime, the Fermat-Euler theorem implies that:

$$a^{(p-1)} \equiv 1(mod\, p)$$

Consider the case of $N = pq$, in which N is a product of exactly two primes p and q. Here, $\phi(N) = (p-1)(q-1)$. To see why $\phi(N)$ is $(p-1)(q-1)$, observe that integers smaller than $N = pq$ that divide N is exactly the set $\{1, p, 2p, \ldots (q-1)p\} \cup \{1, q, 2q, \ldots (p-1)q\}$, which has $pq - p - q + 1$ $= (p-1)(q-1)$ distinct integers.

Intuitively, this statement says that if I take any element of the group $\mathbb{Z}_N^*$ or $\mathbb{Z}_p^*$, and perform the group operation on it "the order" times, the result will be congruent to 1 (i.e. we cycle back to 1).

Easy Problems. In modular arithmetic, several problems have known fast solutions. We briefly mention these problems which are known to be easy to solve for large inputs (even of size 1024 bits or more).

[2] Two integers x and y are co-prime if they don't have non-trivial factor in common, or equivalently $\mathsf{gcd}(x, y) = 1$.

- *Computing GCD.* The Euclidean algorithm can be used to find the gcd (x,y). It was invented in 300 BC, and is considered to be one of the oldest algorithms in use.
- *Finding (x,y) in $ax+by = \mathsf{gcd}(a,b)$.* Given (a,b), it is known how to efficiently find two integers (x,y), such that $ax+by = \mathsf{gcd}(a,b)$, using the *extended Euclidean* algorithm.
- *Finding inverses in $\mathbb{Z}_N^*$.* Finding an inverse of $a \in \mathbb{Z}_N^*$ is equivalent to finding a pair of integers (a^{-1}, y) such that $a \cdot a^{-1} + y \cdot N = 1$. The $\mathsf{gcd}(a,N)$ is 1 since all elements of $\mathbb{Z}_N^*$ are co-prime to N, and the solution to the identity above can be found using the extended Euclidean algorithm as stated above.
- *Checking Primes.* Given an number n, deterministically checking if it is a prime is efficiently possible with AKS algorithm proposed in 2000. Probabilistic algorithms that succeed with high probability are known from earlier.
- *Solving a system of linear congruences.* Given integers (p,q) such that $\mathsf{gcd}(p,q) = 1$, integers (a,b), and a system of simultaneous congruences $x \equiv a (mod\, p)$ and $x \equiv b (mod\, q)$, finding x that satisfies the congruences is efficiently computable using the *Chinese remainder theorem.* The method extends to arbitrary number of congruences.

Hard Problems. Some problems are known to be hard to solve in general. We do not know of efficient solutions to these problems for large (say 1024-bit) (suitably chosen) inputs. There are two hard problems we will use in building public-key cryptosystems.

- *Discrete Logarithm.* Given two elements y and g of a group $\mathcal{G}$, the discrete log problem is to find x such that $g^x = y$. For a large prime p, the discrete log problem is hard in the group $\mathbb{Z}_p^*$ for certain "safe" primes p [3] . In other words, given (y,g,p) such that $y \equiv g^x (mod\, p)$, finding x is hard.
- *Factoring.* Given a product of two large (unknown) primes p and q, finding p or q is hard.

In this lecture, we will look at two public-key encryption methods utilizing the fact that no efficient solutions are known for the factoring and discrete logarithm problems.

2 ElGamal Encryption

The ElGamal encryption system was invented in 1984. It relies on the hardness of the discrete logarithm problem. The initial Setup requires Alice and Bob to agree on public parameters consisting of a large prime p and a primitive root (or generator) g of group $\mathsf{QR}(\mathbb{Z}_p^*)$ [4]. Bob selects a *private* key a, and generates his public key $\beta \equiv g^a \ (mod\, p)$. Bob publishes (g,p,β) publicly. Alice can communicate securely with Bob using this public information, using the following Encrypt procedure:

1. Alice chooses a random secret k and computes $r \equiv g^k (mod\, p)$.
2. Alice computes $t \equiv \beta^k \cdot m \ (mod\, p)$
3. Alice sends the ciphertext (r,t) to Bob

Bob decrypts using his private key a and the received ciphertext using the Decrypt function defined as $m \equiv t \cdot r^{-a} \ (mod\, p)$.

[3]Primes chosen that $p = 2q+1$ where q is a large prime are called "safe" primes. If $p-1$ has small prime factors, fast algorithms to solve discrete log such as the Pohlig-Hellman algorithm are known, and such primes are "unsafe" for use here. In short, primes have to be chosen safely to ensure that discrete log is hard $\mathbb{Z}_p^*$.

[4]This group is the sub-group of quadratic residues in $\mathbb{Z}_p^*$, where $p = 2q+1$, with p and q as primes. The generator of this sub-group is ℓ^2 if ℓ is generator of $\mathbb{Z}_p^*$

Let us check that the three procedures $(\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ form a valid public-key encryption system. For all messages $m \in \mathsf{QR}(\mathbb{Z}_p^*)$, is $\mathsf{Decrypt}(\mathsf{Encrypt}(m)) \equiv m$? This is true for $m < p$, as:

$$t \cdot r^{-a} \equiv \beta^k \cdot m \cdot (g^k)^{-a} \equiv (g^a)^k \cdot m \cdot (g^k)^{-a} \equiv (g^{ak}) \cdot m \cdot (g^{-ak}) \equiv m \,(mod\, p)$$

Security Argument. The first observation is that the public key β leaks nothing about the private key a, even given g and p. This is because finding the private key a given $g^a(mod\, p)$ is the hard discrete log problem. By the same argument, Alice's secret parameter k chosen in Step 1 of the $\mathsf{Encrypt}$ procedure is hidden; neither r nor t reveal anything about k since it is used in the exponent. In other words, to the adversary r and t appear to be two random elements in $\mathsf{QR}(\mathbb{Z}_p^*)$ without knowing a. The second thing to ask yourself is "why did we need k"; indeed, you can check what can go wrong if you directly use β in the step 2 (treating $k = 1$ effectively) to encrypt one message. The encryption is insecure because given β, computing β^{-1} is easy — we can multiple t with β^{-1} to recover the original message is k were always set to 1. In fact, even if k were a secret, if it were deterministically chosen for all message encryptions, it would lead to an insecure encryption method since two messages with the same plaintext value would yield the same ciphertext. Therefore, k must be something that is secret as well as randomly chosen for each message encryption. It acts something like a "blinding factor", which multiplies the original message with a random group element each time.

A more rigorous proof of semantic security is based on a cryptographic hardness assumption called the Decisional Diffie-Hellman (or DDH) assumption. We will see how DDH is closely related to the discrete log problem shortly. We will define DDH hardness assumption for the multiplicative group $\mathsf{QR}(\mathbb{Z}_p^*)$, on which the DDH assumption holds to date. For brevity, we eliminate the $(mod p)$ operation for each multiplication — therefore, when you see something like g^x below, it is implicit that the group operation is multiplication with reduction $(mod\, p)$ and that the operation is repeated x times on the generator. We define the DDH hardness assumption as follows:

Definition 2.1. (Decisional Diffie-Hellman Assumption) Given a cyclic group $\mathsf{QR}(\mathbb{Z}_p^*)$ of prime order q and a generator g, the triples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable all efficient adversaries, for parameters a, b, c chosen uniformly at randomly from the set $\{1, 2, \ldots, q\}$.

Put simply, if the adversary were given (g^a, g^b), and asked to distinguish between g^{ab} from g^c for a random c, without knowing a and b, the DDH assumption says that the adversary would fail. The adversary, by the DDH assumption, cannot distinguish g^{ab} from a random element of G.

The DDH assumption implies that the discrete log problem is hard. To see why, you can readily see that if discrete log is easy, then the adversary can recover a from g^a and b from g^b; then computing g^{ab} is easy and distinguishing it from a random g^c is easy too. Thus, DDH is an assumption that is stronger than discrete log.

Now, we will show that breaking ElGamal's security is equivalent to solving DDH on $\mathsf{QR}(\mathbb{Z}_p^*)$. Define $\mathsf{ElGamal\text{-}Security}$ to be problem that given the encryptions of $E(m_o)$ and $E(m_1)$, determining which one corresponds to m_0. Specifically, we give the adversary the public key (g, p, β) and two ciphertexts (r_0, t_0) and (r_1, t_1), and ask it to determine which ciphertext is a valid encryption of m_0 and m_1 respectively. The adversary is not given the private key a which is chosen at random. We will show that $\mathsf{ElGamal\text{-}Security} \Rightarrow \mathsf{DDH}$. The proof is by reduction.

Proof. We will first show that if there is an efficient adversarial algorithm that solves DDH on $\mathsf{QR}(\mathbb{Z}_p^*)$, then there exists an an attack on $\mathsf{ElGamal\text{-}Security}$. Assume there exists an algorithm called $\mathcal{A}_{\mathcal{DDH}}$ that given pairs (g^x, g^y, g^{xy}) and (g^x, g^y, g^c) (for a random c), it can distinguish the two, thereby violating the DDH assumption. We will show how to create a procedure $\mathcal{A}_{\mathcal{E}}$ using $\mathcal{A}_{\mathcal{DDH}}$ that solves $\mathsf{ElGamal\text{-}Security}$. $\mathcal{A}_{\mathcal{E}}$ is simple — it gives the $\mathcal{A}_{\mathcal{DDH}}$ algorithm two triples, $(\beta, r_0, t_0 \cdot m_0^{-1})$ and $(\beta, r_0, t_0 \cdot g^r)$, where (r_0, t_0) is the first ciphertext and r is chosen randomly. The response from $\mathcal{A}_{\mathcal{DDH}}$

distinguishes the two inputs iff (r_0, t_0) are a valid encryption of m_0; thus $\mathcal{A}_\mathcal{E}$ can distinguish the encryption of m_0 from that of other messages such as m_1. $\square$

A similar reduction can be done for the reverse direction, showing $\mathsf{DDH} \Rightarrow \mathsf{ElGamal\text{-}Security}$. We will not do the proof here. The two statements establish that $\mathsf{ElGamal\text{-}Security} \Leftrightarrow \mathsf{DDH}$.

Remark. The presented ElGamal encryption system is shown to be semantically secure under what is known as the *chosen-plaintext attack* (CPA). In this attack model, the adversary gets to choose which plaintexts it sees the encryption for, but doesn't get to see the decryptions of encrypted ciphertexts of its choice. We could indeed consider a stronger adversary model, where the attacker gets to see both encryptions of its chosen plaintexts as well as decryptions of ciphertexts it chooses. This latter model is called a *chosen-ciphertext attack* or CCA model. Specifically, the CCA-game works by the adversary given a challenge ciphertext c to break as a first step. Then, the adversary can see encryptions of messages of its choice, as well as decryptions of ciphertexts other than the challenge ciphertext c. It's goal is to guess the plaintext of the challenge ciphertext c, with success probability better than that of a random guess.

It can be shown that ElGamal is insecure against the CCA model. There is something subtle here — it turns out that given only the encryption of $E(m)$ and $E(x)$ (not knowing x and m itself), the adversary can easily compute $E(x \cdot m)$! How? Try to multiply the (r, t) ciphertext pairs for two distinct messages and see what ciphertext you can get as an exercise. Now, how should a CCA attacker utilize this "feature" of the ElGamal encryption system? Well, it is not allowed to directly query for the decryption of the challenge ciphertext c. To attack, it picks a random message m, obtains $E(m)$, and then queries for the decryption of $E(m) \cdot c$ instead. The adversary thus obtains $m \cdot D(c)$. By multiplying this final value with m^{-1} it can recover the decryption of c (or $D(c)$). Thus, ElGamal is insecure under a CCA adversary.

Finally, note that the DDH assumption can hold on groups other $\mathsf{QR}(\mathbb{Z}_p^*)$ too. On such groups, one can build an ElGamal encryption system. In fact, there is another popularly used group in cryptosystems today called an "elliptic curve"; it is possible to build the analogous ElGamal scheme on such a curve as the base group.

3 RSA

RSA is another popularly used public-key encryption system. RSA relies on the hardness of *factoring* large numbers. Suppose we are given a 2048-bit number N and told that it a product of two large (e.g. 1024-bit) prime numbers p and q. Can you factor N and recover p and q with any reasonable amount of computation effort? To date, no efficient algorithms are known for this problem. Some integers are indeed easy to factor; however, the problem is hard in general, and in practice we use a class of well-chosen product of primes.

In this encryption system, we are utilizing the mutiplicative group of integers that are co-prime to N, called $\mathbb{Z}_N^*$,where $N = p \cdot q$ and the group operation is integer multiplication modulo N. We will omit the "*mod N*" for brevity, but know that all multiplications are followed by a reduction *mod N* operation in the following construction.

The RSA procedures begins with a trusted setup step run once by Alice and Bob secretly. The Setup function picks two large primes p and q at random, and publishes $N = p \cdot q$. After this, Bob picks a private key $d \in \mathbb{Z}_N^*$, and computes the public key parameter e such that $e \cdot d \equiv 1 (mod\, \phi(N))$. The pair (e, N) is published as the public key for Bob. The adversary does not know p or q, nor does it have access to Bob's private key d.

Alice can $\mathsf{Encrypt}$ a message m to Bob as follows:

$$E(e, N, m) := m^e (mod\, N)$$

Bob can Decrypt a the ciphertext c as follows:

$$D(d, N, c) := c^d (mod\, N)$$

Validity of RSA. Let us check that (E, D) are a valid encryption and decryption function pair — that is, $E(e, N, D(d, N, m)) = m$ for $0 < m < N$. We proceed as follows

$$c := m^e (mod\, N)$$

$$D(d, N, c) := c^d (mod\, N) \equiv (m^e)^d (mod\, N) \equiv m^{ed} (mod\, N)$$

We observe that since $e \cdot d \equiv 1(mod\, \phi(N))$, there must exist some integer k such that $e \cdot d = k \cdot \phi(N) + 1$. Now, we can proceed with the analysis above as follows

$$D(d, N, c) := m^{ed} (mod\, N) \equiv m^{k \cdot \phi(N)+1} (mod\, N) \equiv m^{\phi(N) \cdot k} \cdot m(mod\, N)$$

Using Fermat-Euler theorem, we simplify $m^{\phi(N)}$ term to 1 as follows:

$$D(d, N, c) := (m^{\phi(N)})^k \cdot m(mod\, N) \equiv 1^k \cdot m\ (mod\, N) \equiv m\ (mod\, N)$$

Hence, we establish that $D(d, N, c)$ is m.

Security of RSA. The hardness of breaking RSA relies on keeping (p, q) hidden from the adversary. The RSA problem is no harder than factoring large N. To see why, we will show that if you can factor N to recover (p, q), RSA is insecure. If the adversary were to recover p and q, it could easily compute $\phi(N) = (p - 1)(q - 1)$. Knowing that $e \cdot d \equiv 1(mod\, \phi(N))$, and the public key (e, N), the attacker can recover d by using the extended Euclidean algorithm. After all, d is the inverse of e modulo $\phi(N)$.

You may wonder if the attacker has any additional source of advantage beyond knowing (p, q) in breaking RSA. Is factoring strictly harder than breaking RSA? Or, are the two problems equivalent? It turns out that breaking RSA is equivalent to factoring; however, this is harder to show. Divesh Aggarwal, who is now a professor at NUS, along with Ueli Maurer proved this result in 2008.

To check your understanding of RSA security, ask yourself the following questions:

- If the adversary knows N and p, can it get q?
- If the adversary knows p, can it compute $\phi(p)$?
- If the adversary knows p and q, can it compute $\phi(N)$?
- Given N, can the adversary directly compute $\phi(N)$?

Integrity & Authenticity

Message Authentication Codes and Digital Signatures

CS3235 Computer Security — Prateek Saxena

Encryption guarantees secrecy, but not integrity. It is a common mistake in designing crytographic systems to treat encryption as an integrity mechanism. In this lecture, we will look into mechanisms to protect integrity of messages in two familiar settings of symmetric-key and public-key cryptography.

1 Message Authentication Code (MAC)

A MAC is a cryptographic primitive thats allows checking the integrity of messages and ensuring that they are sent by the intended party. The primitive is used in a symmetric-key setting, wherein Alice and Bob have a pre-established secret key k. A *message authentication scheme* (or MAC) is a pair of algorithms $(\mathcal{S}, \mathcal{V})$, called the signing and verification functions respectively. The function $\mathcal{S} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ is a function that takes a secret key and a messsage as input, and produces a *tag* as output. The function $\mathcal{V} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \rightarrow \{0, 1\}$ takes a secret key k, a message m and a tag t as inputs, and outputs 1 (or `true`) iff $\mathcal{S}\ (k, m) = t$, otherwise outputs 0 (or `false`).

A valid tag for a message allows a party to verify two properties: (a) *authentication*: that the message was produced by some party that knows the secret key, and (b) *integrity*: that the message being checked is exactly the original message for which the tag was generated. It is easy to see how Alice and send a message to Bob, along with its MAC tag, and Bob can verify its integrity and authenticity using the function $\mathcal{V}$.

The security of a MAC scheme $(\mathcal{S}, \mathcal{V})$ is characterized by a game between the adversary and a challenger. The challenger (e.g. Alice) selects a secret k uniformly at random, which is unknown to the adversary. The adversary can ask the challenger to sign a large set of messages $X = \{m_1, m_2, \ldots, m_q\}$ and obtain their corresponding tags $Y = \{t_1, t_2, \ldots, t_q\}$. The adversary can pick the message m_i in the game adaptively by seeing the tags for previous messages $\{1, \ldots, m_{i-1}\}$. Then, if the adversary can forge or construct any message-tag pair (m, t), different from the ones it had observed in the game (i.e. $(m, t) \notin X \times Y$) , such that $\mathcal{V}(m, t)$ is `true` with significant probability, the MAC scheme is insecure. Otherwise, the MAC is said to be secure against *existential forgery* [1]. In short, a MAC scheme is secure if there doesnot exist any message for which an efficient adversary can forge a valid MAC tag by seeing the tags for other messages.

Let us consider how to construct a MAC using some of the symmetric-key primitives we have seen. We start with a cryptographic hash function h. Observe the $h(x)$ achieves the properties of a MAC partially. Specifically, given $h(x)$, it is difficult to find an x' for which $h(x) = h(x')$, if h is collision-resistant. This achieves the integrity property. However, it does not achieve authentication. When Bob recieves a message with a MAC tag from Alice, he cannot be sure that the MAC is generated by Alice because the secret key is not used in the hash computation at all. Indeed, anyone with the knowledge of message and the hash function can create a MAC tag. Thus, this scheme does not satisfy the security goals of a MAC.

[1]The security property is existential forgery under an adaptively chosen message attack

1.1 Constructions from Block Ciphers

We show how one can use a block cipher, also called a pseudorandom function (PRF), as a MAC scheme. Let $F : K \times X \to Y$ be a secure PRF, as defined in previous lectures. The signing function of the MAC scheme $\mathcal{S}$ (k, m) is simply $F(k, m)$. The verification function $\mathcal{V}(k, m, t)$ is the equality check $F(k, m) == t$. The reason why a PRF acts as a good MAC is that if the key is randomly chosen, the resulting function $F(k, \cdot)$ acts like a random function from the message space to the tag space. A random function is very likely to be collision-free; indeed, the probability of collisions is the birthday problem and we have seen that if the output space is large enough, the probability of collisions is negligible. Therefore, given $F(k, m)$, it is difficult to find another message $m' \neq m$ that produces $F(k, m') = F(k, m)$. Further, only someone with the knowledge of the key k can produce a valid tag, since the function $F(k, \cdot)$ is completely different from $F(k', \cdot)$, for $k \neq k'$, by definition of a PRF.

We have seen that if we wish to create a MAC for one block message, we can use a secure PRF or what is called called a block cipher. How do we create a MAC for a larger message? We now present a naive construction, which is insecure and doesn't meet the definition of existential forgery. The construction is analogous to the insecure ECB mode studied in the context of encryption. It breaks the large message m into smaller blocks $\langle m_1, m_2, \ldots m_q \rangle$, such that each block is the size of the input to the block cipher (e.g. 128 bits for `AES`). The signing function the concatenation [2] of the block cipher outputs, written as $\mathcal{S}$ $(k, m) := F(k, m_1)||F(k, m_2)|| \ldots F(k, m_q)$. The verification function checks each block for equality.

Why is the naive construction insecure? There are several attacks possible. First, the adversary can swap the internal blocks of a message, to obtain a valid tag of a different message. Specifically, $F(k, m_1)||F(k, m_2)$ can be constructed from the tag $F(k, m_2)||F(k, m_1)$ by swapping output blocks; thus the adversary can construct a valid tag for the message $m_1||m_2$ by seeing the tag for $m_2||m_1$, violating existential forgery. A second attack called the length extension attack is discussed below, which is more general as it applies to other constructions that handle variable length messages too.

Length Extension Attack. In a length extension attack, the attacker can create $\mathcal{S}(m_1||m_2)$ by obtaining tags for m_1 and m_2. In the naive construction above, the length extension attack is simple to forge as $\mathcal{S}(k, m_1||m_2)$ is $\mathcal{S}(k, m_1)$ $||$ $\mathcal{S}$ (k, m_2). For instance, given the MAC of messages "`Alice did`" and " `pay Bob`", the adversary can easily construct a valid tag for the message "`Alice did pay Bob`" without knowing the secret key.

Raw CBC-MAC. Consider this construction shown in the Figure 1, which is called the raw CBC-MAC construction using a secure block cipher E. The construction divides the message m into blocks $\langle m_1, \ldots, m_q \rangle$, each of the a given block length, and computes $c_i = E(k, m_i \oplus c_{i-1})$ for $i \in [1, q]$. The value c_0 is called the initialization vector (IV), and the final block output c_q is the MAC tag. Unlike the CBC mode used in encryption, where the IV must be randomly chosen, the IV here must be fixed (e.g. to zero). It need not be secret, just that the adversary should not get to tamper / manipulate it.

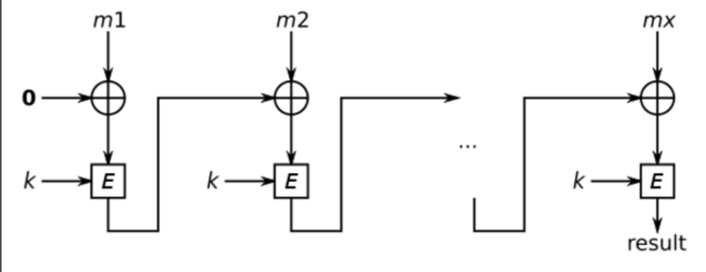

Figure 1: Raw CBC-MAC construction

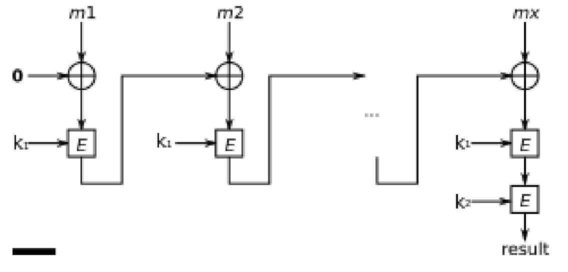

Figure 2: ECBC-MAC construction

[2]The concatentation of bitstrings is denoted by $||$.

Length Extension Attack on Raw-CBC. Raw CBC-MAC is secure for fixed-length messages. However, if you permit variable-length messages, it is possible to perform an attack that is similar to a length extension attack. Specifically, if the adversary gets message-tag pairs (m, t) and (m', t'), where say m' is one block long, it can construct a valid tag for another message m'' for which the tag will also be t'. Let m'' be the message $m||(m' \oplus t)$; notice that it will have the raw-CBC-MAC tag of t', which is the same as the tag for m' creating a forgery attack. To see why, observe that the input to $E(k, \cdot)$ for the first block after m in m'' is $(m' \oplus t \oplus t) = m'$. Thus, the output $E(k, m') = t'$. The message (m'', t') is now a new valid (forged) message-tag pair through the length extention attack.

CBC-MAC. There are several ways to fix the above construction. One could prepend the length of the message to avoid length extensions. If the length of the message is not known in advance, a simple mechanism is to add a final encryption step to terminate the chain, as shown in Figure 2. This construction is called a ECBC-MAC and is known to be secure.

1.2 Other constructions

Several other constructions for MACs are used in practice, such as HMAC, PMAC, NMAC and so on. HMAC is a popular choice and is considered relatively easy to implement without subtle errors; CBC-MAC has been found to be implemented wrongly often in the wild. HMAC is also secure against length extension attacks.

Finally, if you wish to use something that both encrypts and MACs a message, consider using one of the standard *authenticated encryption* techniques. Authenticated encryption (or AEAD) is secure way of both encrypting and MACing a message. Construction providing AEAD modes such as CCM and GCM are available in off-the-shelf cryptographic libraries.

2 Digital Signatures

We now consider the setting where Alice and Bob don't share a pre-established secret, but instead publish their asymmetric public keys and keep their private keys secret. Digital signatures are the equivalent of the MAC in the public-key setting. A signature scheme is a pair of algorithms $(\mathcal{S}, \mathcal{V})$, where the signing function $\mathcal{S}$ takes a private key and a message as input, and outputs a digital signature (equivalent to a tag in MAC). The verification function $\mathcal{V}$ takes the public key, the message, and the digital signature to verify if the signature is constructed from the corresponding private key on the message. The existential forgery definition for MAC extends naturally to signatures. Essentially, the secret key in a MAC is replaced by a public-private key pair in this setting; the private key is used to sign a message and the public key is used to verify.

2.1 Textbook RSA signatures

The RSA encryption system has a natural digital signature mechanism associated with it. Recall that RSA encryption scheme has a public key (e, N) and private key (d, N), for $N = pq$ where p and q are large and suitably chosen primes. The parameters e and d are inverses of each other $mod\, \phi(N)$, or equivalently $e \cdot d \equiv 1\, (mod\, \phi(N))$. The scheme relies on the adversary not knowing d, p, q, and $\phi(N)$, and on the assumption that factoring N is hard.

The textbook RSA signing method generates a signature c for a message m is defined as $\mathcal{S}\ (d, N, m)$ $:= m^d\, (mod\, N)$. The textbook RSA verification method is defined as $\mathcal{V}\ (e, N, c, m) := (c^e\, (mod\, N) == m\, (mod\, N))$.

Validity of RSA Signatures. Let us check that $(\mathcal{S}, \mathcal{V})$ are a valid signing and verification function pair — that is, $\mathcal{S}\ (d, N,\ \mathcal{V}\ (e, N, m), m) = \texttt{true}$ for $0 < m < N$. We proceed as follows

$$c := m^d (mod\, N)$$

$$c^e (mod\, N) \equiv (m^d)^e (mod\, N) \equiv m^{ed} (mod\, N)$$

We observe that since $e \cdot d \equiv 1 (mod\, \phi(N))$, there must exist some integer k such that $e \cdot d = k \cdot \phi(N) + 1$. Now, we can proceed with the analysis above as follows

$$m^{ed} (mod\, N) \equiv m^{k \cdot \phi(N)+1} (mod\, N) \equiv m^{\phi(N) \cdot k} \cdot m (mod\, N)$$

Using Fermat-Euler theorem, we simplify $m^{\phi(N)}$ term to 1 as follows:

$$(m^{\phi(N)})^k \cdot m (mod\, N) \equiv 1^k \cdot m\ (mod\, N) \equiv m\ (mod\, N)$$

Hence, we establish that $\mathcal{S}\ (e, N,\ \mathcal{V}\ (d, N, m), m) = \texttt{true}$.

Security Analysis. Is the RSA signature scheme above secure against existential forgery? Specifically, given the signatures of a few messages, can the adversary exhibit a valid signature for *any* other message. Notice that this definition of security is quite liberal and allows the adversary complete freedom in choosing which message it wishes to forge a signature for. In many previous definitions (e.g. chosen plaintext attack), we gave the adversary a specific challenge c and asked it to break it.

The answer is that textbook RSA is not secure, on closer inspection. We will see a couple of attacks. First, the pair $(1, 1)$ is a valid message-signature pair. To check this, observe the $1^d\ (mod\, N)$ is 1, irrespective of what the private key d is. The attacker doesn't need anything beyond the knowledge of the RSA encryption scheme to exhibit this "attack". For a second attack, observe that $\mathcal{S}(Pr_k, m_1) \cdot \mathcal{S}(Pr_k, m_2)$ is $\mathcal{S}(Pr_k, m_1 \cdot m_2)$, since ${m_1}^d \cdot {m_2}^d = (m_1 \cdot m_2)^d$. Therefore, the adversary can obtain the signatures of m_1 and m_2 and readily exhibit the signature for a different message m_2.

2.2 Practical RSA Signatures

One way to solve the above issues with textbook RSA is to slightly modify the scheme. Instead of using messages directly, if we first hash them, the resulting scheme is secure. Specifically, the secure RSA signing method that generates a signature c for a message m is defined as $\mathcal{S}\ (d, N, m) := h(m)^d (modN)$. The textbook RSA verification method is defined as $\mathcal{V}\ (e, N, c, m) := (c^e\ (modN)\ == h(m)\ (modN))$. The function h is assumed to be perfectly secure, in the sense that it acts as a random oracle.

Security Analysis. Let us revisit the security of this modified RSA signature scheme, called practical RSA signature. Observe that the two attacks outlined above are defeated. The pair $(1, 1)$ is no longer a valid message-signature pair, since the $h(1)$ is not going to be 1 with high probability. Similarly, multiplying the signatures of two messages results in multiplying their hashes; the resulting value is not a predictable function of the messages. In effect, a secure hash function destroys all algaebric structure in the scheme, making it hard to exploit any known relationship between $h(m_1)$ and $h(m_2)$.

The remaining security argument hinges on the one-wayness of the RSA function. Specifically, it is assumed that the problem $\mathsf{RSA}^{-1}_{N,e}$ of finding $m \in \mathbb{Z}^*_N$ given $\langle m^e\ (mod\, N), e, N \rangle$ is hard to solve without knowing d. You are asked to compute the e^{th} root of m^e in $\mathbb{Z}^*_N$, given e and N but not d. This problem is hard if the factors of N are not known, for appropriately chosen e and N. In fact, finding e^{th} roots modulo N is equivalent to factoring N. The difficulty of inverting the RSA function makes it hard to find any message m, such that signature of m maps to a signature chosen by the adversary.

The analysis presented here is not rigorous and beyond this scope of this class. To make it rigorous, we would need to show that if we forge a signature for any message by observing other messages, that would tantamount to either solving the $\mathsf{RSA}^{-1}_{N,e}$ problem efficiently or breaking the cryptographic guarantees of the hash function used.

Practice. The PKCS#1 standard defines constructions based on RSA signatures. It supports a family of hash functions and there are several details about how to map a hash function output (which is of a smaller size say 256 bits) to a message in the $\mathbb{Z}_N^*$ space (where N is 1024 or 2048 bits) which have skipped here.