Lecture Notes on "Introduction to Modern Cryptography"

Yuzhe Tang
Syracuse University, NY, USA, Email: ytang100@syr.edu
NOVEMBER 11, 2021

CONTENTS

I. Modern cryptography: an informal introduction

Crypt-o-graphy, as defined in the dictionary, is "the art of writing or solving codes." In Greek, crypt(o) means "secret", graphy means "writing", logy (as in cryptology) means "study". This dictionary definition only applies to classic cryptography.

Modern cryptography is different from classic cryptography in two senses: 1. classic cryptography as an art versus modern cryptography as a science (in a formal treatment), 2. classic cryptography is used in military settings, while modern cryptography is used in daily life.

The focus of this course is *formal study of modern cryptography*. To start with, let's give an informal introduction to the topic. In the following, we will set up the stage by introducing Private Key Encryption, and focus on introducing the formal treatment of security in modern cryptography.

A. The case of private key encryption or SKE (§ 1.2 in [1];) 08/29/19

A motivating scenario: Hosting files in Dropbox: User Alice stores her personal photo on Dropbox.com. User Bob, being a friend of Alice, gets the photo by downloading it from Dropbox.com. Eve, a thief, wants to get a copy and sell the photo.

Model of SKE: The model of private-key encryption is three parties: Alice sends messages to Bob, and Eve can eavesdrop the communication between the two users.

An encryption scheme is historically called cipher, codes. An encryption scheme, denoted by Π, is defined by key k, message m, ciphertext c, and three algorithms for key generation, encryption and decryption. Formally, a private-key encryption Π (GEN,ENC,DEC) is defined by $k \leftarrow \text{GEN}, c \leftarrow \text{ENC}(k, m), m := \text{DEC}(k, c)$.

What is private in the scheme to Alice/Bob includes: key, message. What is released in public (exposed to Eve) includes: ciphertext, and the three algorithms GEN,ENC,DEC. *Kerckhoff's principle* (in 19th century) states "the cipher method must not be required to be secret" (easy to fall into the hands of enemy). In modern-crypt terms, it means the three algorithms are public. There are three reasons (practical reasons) behind the principle: easier to keep the secrecy of shorter keys than that of an algorithm, easier to change keys upon leakage than changing algorithms, better to have the same encryption algorithm for all users than custom algorithms. Cryptography emphasizes the use of *standard* cryptographic algorithms which have gone through rigorous reviews. It is strongly against the design to use any "home-brewed" encryption algorithms! Note that Kerckhoff's principle implies the security fully relies on *key secrecy*, with algorithms being public.

Security formalization:

Correctness: Given a message, encrypting the message and decrypting the ciphertext should produce the original message. Given an SKE scheme Π and key k, the correctness of Π is that for any m, it has to hold that $\text{DEC}(k, \text{ENC}(k, m)) \equiv m$.

Message confidentiality: The confidentiality of the message states, intuitively, that other than Alice and Bob, no one including Eve, can know the message. Informally, "In an SKE scheme Π, knowing c,GEN,ENC,DEC, Eve cannot know anything about m and k."

The security goal of an SKE scheme is to state the confidentiality property in a formal and measure-able way. This brings challenges: Security is a negative goal (like one cannot do something), rather than a positive goal (like Alice can do something). To evaluate a negative goal it is much more difficult. To be more specific, consider evaluating the correctness and security. Correctness is that Bob can recover the original message from a ciphertext that Alice encrypts to. It is easy to test the correctness by simply asking Bob to decrypt a ciphertext and to check its equality with the original message. That is, given m,Π (GEN,ENC,DEC), the correctness of Π can be tested by $m \stackrel{?}{=} \text{DEC}(k, \text{ENC}(k, m))$.

However, the security of the same scheme is Eve's inability to recover the original message from a ciphertext that Alice encrypts to. It is much harder to test. The designer must check that all the ways that Eve might be able to recover it are blocked. That is, given m,Π (GEN,ENC,DEC), the security of Π can only be tested by proving that there is no such algorithm, say BROKEN_DEC(), that $c \leftarrow \text{ENC}(k, m); \text{BROKEN\_DEC}(c) = m$. To test this, the designer has to consider all possible algorithms, which have infinite possibilities.

There are **two canonical applications (paradigms)** of private-key encryption: Scenario 1. Secure storage where Alice and Bob are separated by *time*, and 2. secure communication where Alice and Bob are separated by *space*.

In secure storage, e.g., encrypted disk, Alice and Bob are the same user active at different times. The communication channel is the encrypted disk. Eve is someone looking to the disk. Sharing the key (on the same user) is trivial in secure storage.

In secure communication, Alice and Bob are two people in two different places (NY versus CA), Eve is an eavesdropper looking into the communication channel between Alice and Bob (e.g., network). Private-key encryption assumes Alice and Bob have the same private key in the beginning.Sharing a key can be done through off-the-band communication (e.g., Alice flies to Bob to share key). In some cases where off-the-band communication is impossible, sharing the key can be done by key exchanges.

B. Other security goals and cryptographic primitives 08/29/19

For data confidentiality, there are schemes of private-key encryption to do bulk communication (in sessions). We often use public-key encryption to exchange and establish the shared key. We usually use the scheme of certificate and public-key encryption to associate users with (public) keys.

TABLE I: Security properties and cryptographic primitives

Scenarios	Security goals	Primitives	Security assumption (or source of security)
"Record-layer" communication on shared secret keys	Message confidentiality	Private key encryption (SKE)	Info-theoretic hardness
Record-layer communication on shared secret keys	Message integrity	Message authentication code (MAC)	Info-theoretic hardness
Record-layer communication on shared secret keys	Message confidentiality and integrity	Authenticated encryption (AE)	Info-theoretic hardness
Sharing secret keys (hand-shaking)	Confidentiality of secret keys	DH key exchange (DHKE)	Computational hardness
Sharing secret keys (hand-shaking)	Confidentiality and integrity of secret keys	TLS (MAC)	Info-theoretic hardness
Sharing public keys (hand-shaking)	Integrity of public keys shared	PKI (Digital signatures and CA)	Computational hardness and trust to CA

For data integrity, there are schemes of message authentication code MAC to do record-layer communication (in session). We can use digital signatures to ensure the authenticity in key-exchange.

For the support of both confidentiality and authenticity, we can use authenticated encryption.

C. Principles of modern cryptography (§ 1.4) 09/03/19

Provable security is the core principle of modern cryptography. A cryptographic scheme defines the "interface" through which parties interact. A cryptographic construction provides the "function body" of the scheme. Provable security refers to the requirement of a rigorous proof that a construction is secure. Here, there are two core concepts: 1) security definition (§ I-D) that defines what security means in the context of the cryptographic scheme, 2) security assumption (§ I-E). A cryptographic scheme is often constructed based on some unproven assumption about algorithmic hardness.

D. Security definition 09/03/19

As mentioned, the concept of security is about what an adversary cannot achieve (a negative goal). To formally define security, it requires two elements, 1) what is the adversary allowed to do (or *threat model*), 2) in the context of a threat model, what is it the adversary cannot achieve? (i.e., the *security goal*). A threat model states what attacks are in scope or what capabilities an attacker has. The security goal states what the construction guarantees under the attacks in scope.

Next, in the setting of private-key encryption, let's explore the design of a formal security definition. That is, how can one convert "an intuitive idea of security" to a formal security definition?

First, let's consider the case of a single message/ciphertext pair, where Eve is given one ciphertext but tries to guess what the original message is. Note that this the ciphertext-only attack as will be introduced. In this setting, one way to formulate **the security goal** is "no recovery of key", that is, Eve cannot recover key k from the observed ciphertext c. To show the insufficiency of this security goal, a counterexample is $\text{ENC}_1(k, m) = m$ where the key is perfectly protected in $c = m$, as it has nothing to do with key k. However, such encryption is obviously not good, as it discloses the message m. A better security goal needs to include the secrecy of message in it.

Then, one can revise the security goal to be "no recovery of key and entire message." Then, a counterexample is $\text{ENC}_2(k, 1024) = 10**$, which clearly is insecure as it does disclose partial information of the message. A better formulation is "no recovery of any digits in the message," even better, "no recovery of any information about the message," such as whether the number is larger than 1000.

In future lectures, we will talk about how to express this statement, more formally, in a mathematical language.

Second, let's consider what an adversary can do or **the threat model**. In the above analysis, we assumed the "ciphertext-only attack (COA)" where Eve is given only one ciphertext. In practice, Eve may have more capabilities by observing multiple ciphertexts. In addition, Eve can know a common message and observe its ciphertext, such as message "hello" hardcoded in secure-channel software and whose ciphertext can be identified as the very first text exchanged in a handshake protocol. This more demanding attack is called known-plaintext attack or KPA. In COA, Eve knows one or multiple ciphertexts. In KPA, Eve additionally know a *given* pair of plaintext and ciphertext.

More advanced attacks include CPA (chosen-plaintext attacks) and CCA (chosen-ciphertext attacks), which model some real-world attacks to private-key encryption. CPA is the standard attacks that any standard SKE construction has to defend. In CPA, Eve gets to *choose* the message she want to know the ciphertext of. This choice in choosing the message is termed as Oracle access.

Note that a threat model may refer to both what an attack can know about (i.e., attacker's knowledge), and what she can do about it (i.e., attack strategy). An attack strategy is about how Eve can leverage her knowledge to mount a successful attack that breaks the security. Specific attack strategies or algorithms are out of scope to security definition. In general, they are just assumed to be "efficient" algorithms that are not hard. In practice, various attack strategies are used, e.g., frequency attack works by Eve analyzing the frequency of ciphertext to infer the message.

In summary, the security definition consisting of security goal and threat model will be formally described in terms of adversarial games.

E. Precise assumption 09/03/19

Cryptographic constructions are often built on assumptions about computational hardness. For instance, cryptographic hashes are based on the assumption that the best algorithm to find a hash collision is to brute force scan the hash-input space, which is hard. We do not know if there is a better algorithm than brute force, but believe there is none. RSA pubic key encryption, or its security, is based on the assumption that prime factorization is hard or there is no known P algorithm to solve the problem.

The **assumptions** in a cryptographic construction is a statement that is conjectured to be true but is not proven. In general, a construction is secure because the only possible algorithms to break the security are assumed to be high computation complexity (NP). And whether a NP algorithm is really hard to compute relies on the unproven assumption that $P \neq NP$.

Having the assumption explicitly specified is necessary, as it keeps users aware that the assumption may be proven true/false in the future which may affect the security (as in the case of MD5). Also, one can compare different constructions (of the same scheme) built on different assumptions – one should prefer the construction based on a "weaker" and well studied assumption.

F. Real-world security and provable security 09/03/19

Provable security versus real-world security: In a real-world security system stack, application-specific code calls into cryptographic libraries. The application-level security indicates whether the cryptographic primitives are used correctly. For instance, is the choice of CPA-secure scheme proper for the target application (in other words, does CPA capture the real-world attacker's capability in the target application?) For another example, does Facebook.com encrypt users' passwords when storing them on disk? This course is not about application-level security.

Inside the cryptographic library, given the same cryptographic construction, there can be different implementations. The implementation- or system- level security refers to whether a particular implementation of cryptographic construction is secure. Related attacks include side-channel attacks. They are also out of scope in this course.

So the security in scope is the security of cryptographic construction (at the protocol specification layer). Such security is provable in that one can present a formal proof that the construction is secure or satisfying the security definition of the scheme.

In software development cycle, one can consider security after the fact or from the very beginning. Today, mostly security is considered after the fact, a software system is developed and released to be operational first, and is then patched after a security hole is discovered (e.g., by blackhat hackers or whitehat). In the latest wave of software engineering, say for IoT applications, it is advocated that security should be taken as the first class citizen and be addressed before the system is designed.

II. Perfectly-secret encryption

Perfect secrecy is a security definition put down in the language of discrete probability. We give a background introduction of discrete probability as in § A-A. Here, we first present the SKE scheme in the language of probability. Then, we formulate the definition of perfect secrecy.

A. Describing encryption scheme in probability 09/05/2019

Key, message and ciphertext are all modeled as random variables. Formally, a key is denoted by its random variable K, value k and key space $\mathbb{K}$. A message is by random variable M, value m, and message space $\mathbb{M}$. A ciphertext is by random variable C, value c, and space $\mathbb{C}$. Any encryption scheme can be formally described:

$$\begin{aligned} k &\leftarrow \textsc{Gen}(\mathbb{K}) \\ c &\leftarrow \textsc{Enc}(k, m) \\ m &:= \textsc{Dec}(k, c) \end{aligned}$$

Correctness: $\forall k \in \mathbb{K}, m \in \mathbb{M}, \textsc{Dec}(k, \textsc{Enc}(k, m)) \equiv m$

B. Formulating Perfect Secrecy 09/05/2019

1) Perfect secrecy as conditional probability: The security of an encryption scheme can be naturally conveyed by the following: "knowing the ciphertext does not give Eve any additional information about the message." In the above scheme, message is treated as a random variable M. Eve's prior knowledge about the message is modeled by the probability distribution about the event that M takes different values in $\mathbb{M}$. Intuitively, the more the distribution is fixed and is close to the actual value of the message, the more knowledge Eve has. In a ciphertext-only attack, Eve's posterior knowledge about the message is the posterior probability $Pr[M = m|C = c]$. Thus, the additional knowledge Eve obtained through observing the ciphertext in a ciphertext-only attack is $Pr[M = m|C = c] - Pr[M = m]$. The idea of **perfect secrecy** (§ Def-2.3) is that by observing the ciphertext Eve does not gain any extra knowledge about what value a message may take. This idea can be formalized by:

$$Pr[M = m|C = c] - Pr[M = m] = 0 \tag{1}$$

2) Perfect secrecy as perfect indistinguishability: The idea of indistinguishability is to consider a message domain of two values m_0, m_1 and to formulate the security by Eve's inability to distinguish the case that the message (a random variable) takes one value (m_0) from the case that it takes the other value (m_1). By considering two message values m_0, m_1, it simplifies the setting and security analysis. Informally, one can convey the idea of indistinguishability of perfect secrecy as following:

$$\text{Fixing } c, \forall m_0, m_1, Pr[\text{ENC}(K, m_0) = c] = Pr[\text{ENC}(K, m_1) = c] \tag{2}$$

It is proven in the textbook (§ Eqn-2.1) that the above perfect indistinguishability is equivalent to perfect secrecy as conditional probability.

More fundamentally, **perfect indistinguishability** gives us a *standard and generalizable* way of defining security through security games/experiments (for describing threat model) and indistinguishability (describing the security goal).

Security game for private-key encryption, PRIVK, is defined as an interaction between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$:

$\text{PRIVK}^{\text{EAV}}$:

$\mathcal{C}$	$\mathcal{A}$
	$m_0, m_1 \leftarrow \mathbb{M}$
$k \leftarrow \text{GEN}(\mathbb{K})$	
$b \leftarrow \{0,1\}$	
$c \leftarrow \text{ENC}(k, m_b)$	
	$b' \leftarrow \mathcal{A}(c)$

The success of a game is defined to be $b = b'$, namely $\text{PRIVK} = 1$.
Adversarial advantage is $\text{ADV} \stackrel{\text{def}}{=} Pr[\text{PRIVK} = 1] - \frac{1}{2}$

Perfect indistinguishability is defined by requiring a zero adversarial advantage, namely $\text{ADV} = 0$.

C. One-time pad 09/10/19

One-time pad is a simple construction of private-key encryption that satisfies: 1. $\mathbb{K}$ has the same size with $\mathbb{M}$, 2. GEN generates k by following a uniform distribution in $\mathbb{K}$, 3. ENC and DEC algorithms are bitwise XOR.

$$\begin{aligned} c &:= \text{ENC}(k, m) = k \oplus m \\ m &:= \text{DEC}(k, m) = k \oplus c \end{aligned} \tag{3}$$

The preliminary of XOR operation is described in § B-A.

1) Security of OTP: Is OTP perfectly secret? 09/10/19:

Theorem 2.1 (Thm §2.9): OTP is perfectly secure. $\forall c \in \mathbb{C}, m \in \mathbb{M}, Pr[M = m | C = c] = Pr[M = m]$

Proof

$$\begin{aligned} Pr[C = c | M = m] &= Pr[\text{ENC}(K, m) = c] \\ &= Pr[K \oplus m = c] \\ &= Pr[K = c \oplus m] = \frac{1}{2^{|\mathbb{K}|}} \\ Pr[C = c] &= Pr[\text{ENC}(K, M) = c] = \sum_{\forall m' \in \mathbb{M}, \forall k \in \mathbb{K}} Pr[\text{ENC}(k, m') = c] \\ &= \sum_{\forall m' \in \mathbb{M}} \sum_{\forall k \in \mathbb{K}} Pr[\text{ENC}(k, m') = c] = \sum_{\forall m' \in \mathbb{M}} Pr[\text{ENC}(K, m') = c] \end{aligned}$$

Event $\text{ENC}(K, m') = c$ can be expressed in a different form: $Pr[C = c \wedge M = m']$. Therefore,

$$\begin{aligned} Pr[C = c] &= \sum_{\forall m' \in \mathbb{M}} Pr[C = c \wedge M = m'] \\ &= \sum_{\forall m' \in \mathbb{M}} Pr[C = c | M = m'] \cdot Pr[M = m'] \\ &= \frac{1}{2^{|\mathbb{K}|}} \sum_{\forall m' \in \mathbb{M}} Pr[M = m'] = \frac{1}{2^{|\mathbb{K}|}} \end{aligned} \tag{4}$$

Due to Bayes' theorem, $Pr[M = m|C = c] = \frac{Pr[C=c|M=m]}{Pr[C=c]} \cdot Pr[M = m]$

$$\therefore \qquad Pr[M = m|C = c] = \frac{\frac{1}{2^{|\mathbb{K}|}}}{\frac{1}{2^{|\mathbb{K}|}}} Pr[M = m] = Pr[M = m] \tag{5}$$

(Takeaway intuition: $Pr[M = m|C = c]$ is about cracking DEC; given C =c, find a k s.t. M =m. And $Pr[C = c|M = m]$ is about ENC; given M =m, find a k s.t. C =c.)

D. Are OTP and Perfect Secrecy Useful? (Their Practical Limitations) 09/10/19

Real-world application: Red phone: During the Cold War, the "red phone" links US president and USSR chairman using OTP. Stage 1) Before the phone call, a trusted courier working for the US government flies to a preset location to meet a courier working for the USSR government. During the meeting, they shared the "pad" or one-time key which is usually a long string printed on papers in a briefcase. Stage 2) Then, during the phone call, the audio stream is encrypted using the shared pad in the OTP scheme.

1) One-time use of OTP key: An OTP key can only be used for encrypting one message. In other words, OTP with fixed key is not secure in ciphertext-only attacks with multiple messages.

More specifically,

$$\begin{aligned} m_1 \oplus k &= c_1 \\ m_2 \oplus k &= c_2 \\ \therefore m_1 \oplus m_2 &= c_1 \oplus c_2 \end{aligned} \tag{6}$$

Observing c_1 and c_2, Eve can know about $m_1 \oplus m_2$. Further more, observing c_1, c_2 and m_1 (e.g., "hello" message in a handshake), Eve can know about m_2.

2) Key Length:

Theorem 2.2 (Thm §2.10): In any perfectly secure encryption scheme, $|\mathbb{K}| \geq |\mathbb{M}|$.

Proof Fix c, consider a message subspace $\mathbb{M}$ (c) where $\forall m \in \mathbb{M}(c)$, $\exists k \in \mathbb{K}$ s.t. DEC (c,k)=m. Therefore, $|\mathbb{M}(c)| \leq |\mathbb{K}|$.

To prove the theorem by contradiction, assume $|\mathbb{K}| < |\mathbb{M}|$. Thus, $|\mathbb{M}(c)| < |\mathbb{M}|$. That is, $\exists m' \in \mathbb{M} \wedge m' \notin \mathbb{M}(c)$.

$Pr[M = m'] \neq 0 = Pr[M = m'|C = c]$

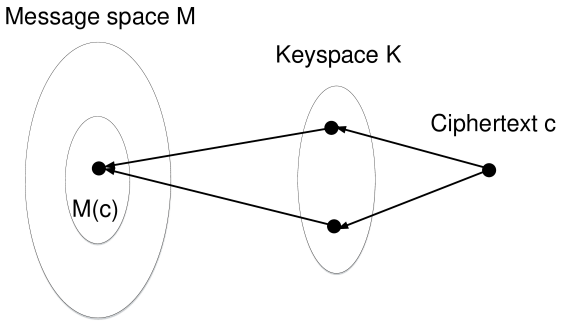

Fig. 1: Perfect secrecy implies a large key space.

The takeaway (or intuition) here is that if $m' \in \mathbb{M} - \mathbb{M}(c)$ exists, Eve observing c can know, with certainty and high confidence, that the original message m (=DEC (k,c)) cannot take value m'.

III. PRIVATE-KEY ENCRYPTION

A. Computational security (§ 3.1)

1) Motivation: Attack a Practical Scheme?: A practical encryption scheme is built on a key space smaller than message space ($|\mathbb{K}| \leq |\mathbb{M}|$).

For such schemes, regardless the specific algorithms in ENC/DEC, it immediately follows that $\exists m' \notin \mathbb{M}(c) \stackrel{\text{def}}{=} \{m|\forall k, m = \text{DEC}(k, c)\}$. Intuitively, knowing c gives the adversary extra information about the message distribution (like the message that c decrypt to cannot be m', although the adversary may not know what m' is). This extra information can be turned into actual attacks.

To design such attacks, we consider KPAattacks (where the adversary has the prior knowledge about some pairs of messages and ciphertexts, say m_i, c_i under the key unknown to her).

First, Eve can mount exhaustive-search attack: 1) In the offline phase, Eve brute-forces the key space $\mathbb{K}$ and for each key encountered say k, she tests if $\forall i, \text{ENC}(k, m_i)? = c_i$. If so, she stops and claims to find the key. 2) In the online phase, Eve uses the found key k to decrypt an observed ciphertext c.

In this exhaustive-search attack, Eve runs an algorithm of complexity $O(|\mathbb{K}|)$ with success rate 100%.

Second, Eve can make one guess by randomly picking one key, k, from the key space and test the same, that is, $\forall i, \text{ENC}(k, m_i)? = c_i$.

In this one-guess attack, Eve runs a constant-time algorithm with success rate $\frac{1}{|\mathbb{K}|}$.

As long as we have a small key space $|\mathbb{K}| \leq |\mathbb{M}|$, the scheme is subject to these two attacks. To design a practical and secure encryption scheme, the idea is to make these two attacks "difficult" to succeed in practice. More specifically, we want to make it hard to exhaust the key space in the first attack. We also want to make the success rate very small (but not equal to zero ...) in the second attack.

More generally, to bypass the limitation of perfect secrecy, we consider two relaxations that are needed: 1. $\mathcal{A}$ is computationally bounded in that she can only run "efficient" algorithms (in the relaxed threat model), and 2. allow a small adversarial advantage (as the relaxed security goal).

2) Formalizing Computational Security: **Security parameter** n is the key concept to formalize the notion of the two relaxations. An efficient adversary can only run a **PPT** algorithm with respect to n as input length to the algorithm. Note that PPT stands for probabilistic polynomial-time algorithms; probabilistic algorithms are use to generate keys and model $\mathcal{A}$'s strategy. Polynomial-time algorithms are those have polynomial-time complexity w.r.t. the input length.

It allows a small adversarial advantage (over the random guess in succeeding the game), formulated as a **negligible function** w.r.t. security parameter, NEGL (n). A function $\text{NEGL}(\cdot)$ is negligible when $\forall x, \forall p(\cdot)$ where $p()$ is a polynomial function, $\text{NEGL}(x) < 1/p(x)$. For example, $\text{NEGL}(x) = 1/2^x$.

Fact: The two relaxations are necessary for the key space to be smaller than message space ($|\mathbb{K}| < |\mathbb{M}|$). Why (Exercise 3.12)

B. Computationally secure encryption (§ 3.2)

Encryption scheme Π (GEN,ENC,DEC) consists of three PPT algorithms:

$$\begin{aligned} k &\leftarrow \text{GEN}(1^n) \\ c &\leftarrow \text{ENC}(k, m) \\ m &:= \text{DEC}(k, c) \end{aligned}$$

$\leftarrow$ (:=) denotes probabilistic (deterministic) assignment.

Correctness: $\forall k \in \mathbb{K}, m \in \mathbb{M}, \text{DEC}(k, \text{ENC}(k, m)) \equiv m$

Security definition(IND-EAV):

The **threat model** of ciphertext-only attack is formulated by running $\text{PRIVK}^{\text{EAV}}(n)$, parameterized with security parameter n (for PPT $\mathcal{A}$) and challenger's choice of bit b.
Game (computational security) $\text{PRIVK}^{\text{EAV}}(n)$

$\text{PRIVK}^{\text{EAV}}(n)$:

$\mathcal{C}$	$\mathcal{A}$
	$m_0, m_1 \leftarrow \mathbb{M}$
$k \leftarrow \text{GEN}(1^n)$	
$b \leftarrow \{0,1\}$	
$c \leftarrow \text{ENC}(k, m_b)$	
	$b' \leftarrow \mathcal{A}(c)$

Successful game: $b = b'$, denoted by $\text{PRIVK}^{\text{EAV}}(n)$.
Security goal: The security of indistinguishability requires that the adversarial advantage $\text{ADV} \stackrel{\text{def}}{=} Pr[\text{PRIVK}(n) = 1] - \frac{1}{2}$ is negligible, or $\text{ADV} \leq \text{NEGL}(n)$.

C. Stronger security notion (CPA) § 3.4

1) Multi-encryption security (09/12/19): **Multi-encryption security** is defined by $Pr[\text{PRIVK}^{\text{EAV-MUL}}(n) = 1] - \frac{1}{2} < \text{NEGL}(n)$ where the game is below:

$\text{PRIVK}^{\text{EAV-MUL}}(n)$:

$\mathcal{C}$	$\mathcal{A}$
	$\{m_{0,1}, \ldots, m_{0,t}\} \leftarrow \mathbb{M}^t$
	$\{m_{1,1}, \ldots, m_{1,t}\} \leftarrow \mathbb{M}^t$
$k \leftarrow \text{GEN}(n)$	
$b \leftarrow \{0,1\}$	
$c_1 \leftarrow \text{ENC}(k, m_{b,1})$	
$\ldots$	
$c_t \leftarrow \text{ENC}(k, m_{b,t})$	
	$b' \leftarrow \mathcal{A}(\{c_1, \ldots, c_t\})$

Theorem 3.1 (Thm §3.2): Any deterministic encryption scheme can not be indistinguishable (secure) under the multi-encryption ciphertext-only attacks.

Proof Consider a concrete game $\text{PRIVK}^{\text{EAV-MUL}}$: 1. $\mathcal{A}$ outputs two message vectors: $\vec{m}_0 = \{0^l, 0^l\}, \vec{m}_1 = \{0^l, 1^l\}$. 2. upon receiving $\vec{c}_b = \{c_0, c_1\}$, $\mathcal{A}$'s strategy is that $b' = 0$ when $c_0 = c_1$; otherwise $b' = 0$. This concrete game always succeed with probability 1 because deterministic encryption will encrypt 0^l and 0^l to the same ciphertext, yet 1^l and 0^l to different ciphertext (otherwise it can't decrypt). That is, $Pr[\text{PRIVK}^{\text{EAV-MUL}} = 1] = 1$.

Message length is assumed to be public information in private-encryption schemes. In practice, message length may be disclosing sensitive information, like an employee salary being a 5-digit number or a 6-digit number or the number of records in a patient database may disclose the patients have cough or cancer. In this case, developers are responsible to pad the messages to the same length.

Here, we use indistinguishability to describe the security. However, the same security can be equivalently defined in a more semantic way, that is, reflecting the intuition that disclosing ciphertext does not disclose any partial information about the message. This is called semantic security. Here, semantic security to indistinguishability is like perfect secrecy to perfect indistinguishability. It is proven semantic security is equivalent to indistinguishability in most encryption/authentication schemes mentioned in the textbook. Indistinguishability is used as the working definition in this course.

2) CPA security (09/17/19): **CPA attacks in the real world**: In a shared terminal, Alice uses the computer with her secret key to send encrypted messages over to the Internet. Alice left without logging herself out. The shared terminal is used by Eve, where Eve can send messages to the encryption engine (algorithm), yet cannot directly observe Alice's secret key.

During the Battle of Midway, Japanese military base encrypted message ("Midway") and telegraphed the ciphertext ("AF") to their battleships. US navy intercepts the ciphertext telegram ("AF") and had an initial guess that "AF" corresponds to "Midway". To confirm the guess (and to convince white house to deploy aircraft carrier), US intelligence sends out a fake message "Water shortage on Midway", intended to be visible to Japanese. By observing "AF" is mentioned in the next round of communication, US intelligence now confirms that "AF" maps to "Midway." In this process, US, the eavesdropper, decided (or influenced) what message got encrypted and can observe the ciphertext.

In both examples, the eavesdropper has the capability in influencing or deciding what message can be encrypted and in observing the corresponding ciphertext. This ability is formalized in the concept of **oracle**. In an encryption oracle, $\mathcal{A}$ invoke ENC () function under message m of her choice, yet $\mathcal{A}$ cannot know the private key k. Formally stated, $\mathcal{A}$ has an "oracle" access to an encryption scheme iff 1) $\mathcal{A}$ can encrypt an *arbitrary* number of messages, $m_2, m_3 \ldots$, and observe their corresponding ciphertexts. The access to the oracle is adaptive in that $\mathcal{A}$ can access it as many times as it wants. 2) $\mathcal{A}$ does not directly observe the private key.

CPA security definition: CPA security (IND-CPA) is defined by $Pr[\text{PRIVK}^{\text{CPA}}(n) = 1] - \frac{1}{2} < \text{NEGL}(n)$ where the game is:

$\text{PRIVK}^{\text{CPA}}(n)$:

$\mathcal{C}$	$\mathcal{A}$
	$m_0, m_1 \leftarrow \mathcal{A}^{\text{ENC}}(\mathbb{M}) \stackrel{\text{def}}{=} \mathcal{A} \leftarrow \{(m_2, c_2), (m_3, c_3), \ldots\} \leftarrow \mathcal{O}(\text{ENC})$
$k \leftarrow \text{GEN}(n)$	
$b \leftarrow \{0,1\}$	
$c \leftarrow \text{ENC}(k, m_b)$	
	$b' \leftarrow \mathcal{A}^{\text{ENC}}(c)$

Facts: CPA (for the single message) is equivalent to CPA for multiple encryption (Thm §3.24).

Fixed-length CPA-secure encryption can be extended easily to result in an arbitrary-length CPA-secure scheme. Given a 1-bit CPA-secure encryption ENC, the arbitrary-length encryption, ENC', is constructed by $\text{ENC}'(m_1\|m_2\|m_3\dots) = \text{ENC}(m_1)\|\text{ENC}(m_2)\dots$. More efficient extension constructions are through the mode of operation (in §3.6).

D. Constructing an encryption scheme §3.3 09/17/19

1) PRG: **Motivation**: How can one test if a string of two bits, say b_1b_2, is randomly selected? One way is to run a series of statistical tests, for instances, 1) if b_1 has 50% chance to take value 1, 2) if b_2 has 50% chance to take value 1, 3) if $b_1 \oplus b_2$ has 50% chance to take value 1, etc. From a security perspective, a truly random string has to pass all possible such statistical tests. In other words, a string is truly random from the eyes (view points) of statistical tester.

PRF definition: A generator $G(\{0,1\}^n) = \{0,1\}^l$ is a deterministic, extension function ($l > n$) that extends a short string to a long string.

A pseudorandom generator $\text{PRG}(\{0,1\}^n) = \{0,1\}^l$ is such a generator that extends a short, truly random string to a longer, random-looking string. The short input is called seed (in space 1^n).

A distinguisher $\mathcal{D}(\cdot)$ is a PPT algorithm that given a long bit string (in space 1^l) determines (or produces a binary decision) if the string is truly random or not.

In one trial, $\mathcal{D}(\cdot)$ takes as input the output string of $\text{PRG}(s)$ with s uniformly distributed. That is, $\mathcal{D}(\text{PRG}(s))$.

In another trial, $\mathcal{D}(\cdot)$ takes as input a bit string r truly randomly (uniformly) picked from the space of 1^l. That is, $\mathcal{D}(r)$.

Informally, $\text{PRG}(\cdot)$ is pseudorandom iff $\mathcal{D}(\text{PRG}(s))$ is indistinguishable from $\mathcal{D}(r)$. That is, it is hard to distinguish the l-bit string of r uniformly picked from space 1^l from the l-bit string produced by $\text{PRG}(s)$ where s is uniformly picked from space 1^n. Formally, the difference of the likelihoods of the two events, $Pr[\mathcal{D}(\text{PRG}(s)) = 1] - Pr[\mathcal{D}(r) = 1]$, is negligible.

Definition 3.2: Generator G is a pseudorandom generator PRG if $\forall$ PPT $\mathcal{D}^{G(\cdot)}(\cdot)$, $Pr[\mathcal{D}(G(1^n)) = 1] - Pr[\mathcal{D}(1^l) = 1] < \text{NEGL}(n)$

Intuition: An exponential-time $\mathcal{D}$ can iterate through the output domain (all 2^l values) and observe if they are concentrated in a very small subspace of the output domain (i.e., $\frac{1}{2^{l-n}}$ of space 2^l). Alternatively, $\mathcal{D}$ checks $r \in S_2$. If so, $\mathcal{D}$ outputs 1 (meaning not random). If the input of $\mathcal{D}$ is PRG (x), $Pr[\mathcal{D}(\text{PRG}(x)) = 1] = 1$. If the input of $\mathcal{D}$ is a random number picked from 1^l, $Pr[\mathcal{D}(r) = 1] = \frac{1}{2}^n$. Thus the probability difference is this: $Pr[\mathcal{D}(\text{PRG}(x)) = 1] - Pr[\mathcal{D}(r) = 1] = 1 - \frac{1}{2}^n$, which is a large percentage close to 1.

In other words, the distribution of pseudorandom numbers in $S_2(\text{PRG})$ is not equal to the distribution of truly random numbers (in $S_1(\text{PRG})$) from the perspective of exponential-time $\mathcal{D}$, but looks sufficiently like the uniform distribution from the perspective of polynomial-time $\mathcal{D}$. .

Discussion: The output domain of a generator G (or extension function) has three interesting sets: $S_1(\text{PRG})$: the output domain itself $\{0,1\}^l$ of 2^l numbers, $S_2(\text{PRG})$: the set of all outputs of G, $\{G(s)|\forall s \in \{0,1\}^n\}$ of 2^n numbers, and $S_3(\text{PRG})$: the observable set of numbers by $\mathcal{D}$ or the oracle of G, $\mathcal{O}(G(\cdot))$ of polynomial numbers w.r.t n. Obviously, $|\{0,1\}^l| > |\{G(s)|\forall s \in \{0,1\}^n\}| > |\mathcal{O}(G(\cdot))|$.

In this setting, G is pseudorandom if the distribution of numbers in $S_2(\text{PRG})$ (generated by G) looks like a uniform distribution in $S_1(\text{PRG})$ from $\mathcal{D}$'s perspective (who can only see $S_3(\text{PRG})$). Formally, "looks like" is formalized by "indistinguishability" as above.

Facts: Generator by appending XOR sum of the bits is not PRG (Example §3.15); $\mathcal{D}$ can be simply checking if $m_l? = m_i \oplus m_2 \dots \oplus m_{l-1}$ which results in 1/2 probability advantage.

2) Construction OTP-PRG*:* : PRG can be used to construct a fixed-length encryption scheme: $k \leftarrow \text{GEN}(1^n)$, $c := G(k) \oplus m$, $m := G(k) \oplus c$. Here, $m, c \in \{0,1\}^{l(n)}$. It is EAV-secure, but not EAV-MUL-secure (deterministic). To make the construct CPA secure, PRG is not sufficient.

E. Constructing CPA-secure encryption (§3.5) 09/19/2019

1) PRF: **A function** f is a mapping: $\{0,1\}^n \to \{0,1\}^n$. A function can be stored in a table of 2^n rows of n-bit output strings. There are $n \cdot 2^n$ bits.

A function generator (or keyed function) is a mapping from a key (of n bits) to a function f. By analogy, a key in PRF is a seed in PRG. A keyed function is $F : \{0,1\}^n(\text{key}) \to f(\text{generated function})$,[1] where A generated function is $F_k : \{0,1\}^n(\text{block}) \to \{0,1\}^n(\text{block})$.

PRF: Intuitively, a function generator is pseudorandom if it maps a short, random key to a long, random-looking function. A "random-looking" function means the following: If the distribution a generated function over the choice of k (i.e., $\text{PRF}_k(\cdot)$) is close enough to the (uniform) distribution of a random function (i.e., randomly picked from the space of $2^{n \cdot 2^n}$ possibilities). In other words, this indistinguishability:

$\text{PRF}(\cdot)$ is a function generated by PRF with random k. $f(\cdot)$ is a random function uniformly picked from its space. $\mathcal{D}$ interacts with a blackbox which can encapsulates either a generated function or a random function. $\mathcal{D}$ interacts with the

[1]This differs the format in textbook.

blackbox through adaptively sending an arbitrary number of input message $\{m\}$ and observing their outputs $\{\mathcal{O}(m)\}$. The distinguisher $\mathcal{D}$ produces a binary result about what is in the oracle is a generated function ($\text{PRF}_k(\cdot)$) or a random function ($f(\cdot)$).

Definition 3.3: A keyed function is PRF if $\forall$ PPT $\mathcal{D}^{\text{PRF}(\cdot,\cdot),\text{PRF}_k(\cdot)}(1^n)$, $Pr[\mathcal{D}^{\text{PRF}_k(\cdot)}(1^n) = 1] - Pr[\mathcal{D}^{\mathbb{F}(\cdot)}(1^n) = 1] < \text{NEGL}(n)$

Fact: One-time pad is not a PRF (details see Example §3.26): OTP $\text{PRF}(k, x) = k \oplus x$ generates function $\text{PRF}_k(\cdot)$ given a key k. Suppose an oracle $\mathcal{O}$ encapsulating either generated function $\text{PRF}_k(\cdot)$ or randomly chosen function $f(\cdot)$. Given this oracle, a distinguisher $\mathcal{D}(x_1, x_2, y_1 = \mathcal{O}(x_1), y_2 = \mathcal{O}(x_2))$ works by returning $x_1 \oplus x_2 == y_1 \oplus y_2 ? 1 : 0$. Thus, $\mathcal{D}^{\text{PRF}_k(\cdot)}(...) \equiv 1$ with 100% probability, and $\mathcal{D}^{f(\cdot)}(...) = 1$ at probability $1/2^n$. (The latter can be worked out by an example with $n = 1$ and x_1, x_2 are one-bit strings). Their difference is $1 - \frac{1}{2^n}$ which is not negligible.

Block cipher/Pseudorandom permutation: A permutation is a bijective function (one-to-one mapping). Its input domain is the same to its output domain. If a PRF generates permutations, it is a pseudorandom permutation generator (or block cipher). It is defined as $\text{PRP} : \{0,1\}^{l_k}, \{0,1\}^l \to \{0,1\}^l$ (l_k is key length). The security of PRP follows PRF.

Block cipher is a term used in practice for PRP with fixed key length.

Facts: PRF can be used to construct PRG and stream cipher; $\text{PRG}(s) = F_s(1) \| F_s(2) \cdots F_s(l)$. Vice versa, PRG can be used to construct a PRF with small block length (polynomial number of rows); result of PRG (k) is interpreted as a table of n rows, thus $\log(n)$ block length.

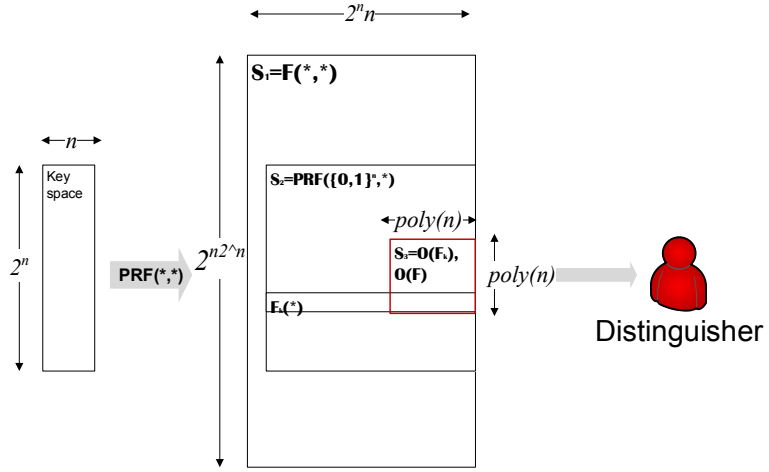

Fig. 2: Intuition of PRF: There are two spaces: a space of 2^n functions called S_1, and a space of $2^{n \cdot 2^n}$ called S_2. S_1 is the space of generated function by $\text{PRF}(\cdot)$ and S_2 is the space of any function mapping n-bit input to n-bit output.A pseudorandom function generator requires the distribution of generated function in S_1 "looks like" the distribution of random function in S_2, through the lens of a PPT distinguisher who can only observes the subset of both S_1 and S_2).

2) OTP-PRF: a CPA-secure construction:

Construction (OTP-PRF): CPA-secure encryption with fixed-length message n:

- $\text{GEN}(1^n)$: choose uniform $k \in \{0,1\}^n$ and output it.
- $\text{ENC}_k(m \in \{0,1\}^n)$: choose uniform $r \in \{0,1\}^n$ and output the ciphertext

$$c := \langle r, c' \stackrel{\text{def}}{=} F_k(r) \oplus m \rangle \tag{7}$$

- $\text{DEC}_k(c = \langle r, s \rangle)$: output

$$m := F_k(r) \oplus c' \tag{8}$$

Correctness can be easily established. Security proof (see Thm §3.31).

Take away points: OTP+PRG is insecure under CPA attacks because of deterministic encryption. In other words, there is only one source of randomness in OTP+PRG. To add another source of randomness (for probabilistic encryption), PRF is a mapping that takes two random numbers as inputs. $\text{PRF}(\cdot,\cdot)$. Using PRF in encryption scheme, the private key is used as the first random input. The second random input is generated each time encryption is invoked.

IV. Modes of operation (09/26/19)

Motivating problem: In real-world scenarios, there are needs to encrypt messages of long and abitrary length. Think about live broadcasting (TV or radio) as an example, where the audio/video message to encrypt is "streamed" and its length is long

and cannot be known in advance. So far, we just talked about fixed-length encryption schemes, like OTP+PRG and OTP+PRF. They cause problems in the scenario with arbitrary-length long messages.

For instance, consider OTP-PRG ($c :=\text{PRG}(k) \oplus m$) where PRG is defined as a mapping from 1^n to 1^l where l is fixed. It requires l or message length to be fixed before ENC is invoked, which is impossible in the streaming scenarios. Supporting streamming message is the problem tackled by stream cipher mode of operations (§ IV-A).

For another instance, consider OTP-PRF ($c \leftarrow \text{PRF}_k(r) \oplus m \| r$) where the ciphertext c has double the length with plaintext m. For a long message, it is not a good news (for a message of 2 GB, this means the ciphertext of 4 GB). Supporting shorter ciphertext is the problem tackled by block cipher mode of operations (§ IV-B).

A. Stream cipher mode of operation

1) Stream cipher: Stream cipher is a specialized PRG generating random numbers of abitrary length. In other words, it provides a continuous source of randomness.

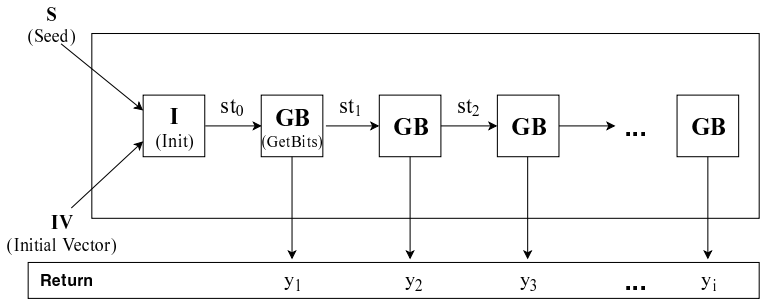

Fig. 3: Stream cipher

The scheme consists of two deterministic algorithms (INIT,GETBITS). The cosntruction is below and is also illustrated in Figure 3.

Algorithm 1 PRGByStreamCipher(seed s, initialization vector IV)

1: $st_0 :=\text{INIT}(s, IV)$
2: **for all** $i \in [1, l]$ **do**
3: $\quad (y_i, st_i) :=\text{GETBITS}(st_{i-1})$
4: $\quad$ **return** $y_1 \dots y_l$
5: **end for**

Fact: In OTP +prg, PRG handles key extension (into the pad) and the mode of operation is about extending fixed-length message to abitrary length.

2) Synchronized mode: **What it is?**: For one message m of message blocks $m_1 m_2 \dots m_i$, it produces a stream of ciphertext blocks $c_1 c_2 \dots c_i$:

$$\begin{aligned} \text{PAD}_1\text{PAD}_2 \dots \text{PAD}_i &:= \text{PRG} - sc(k) \\ c_1 &:= \text{PAD}_1 \oplus m_1 \\ c_2 &:= \text{PAD}_2 \oplus m_2 \\ &\dots \\ c_i &:= \text{PAD}_i \oplus m_i \end{aligned}$$

Use case and why it's called Synchronized mode: Alice and Bob maintain a scynchronized state about what index i it is to encrypt/decrypt the current block of message/ciphertext. Given m_i, Alice needs to know i to get the correct PAD_i for encryption. Given c_i, Bob needs to know i to get the correct PAD_i for descrpytion. It is applicable in scenarios with online Alice and Bob, where Alice and Bob can synchronously maintain this dynamic state. It is also called stateful.

3) Unsynchronized mode: **What it is?**: For message m of message blocks $m_1 m_2 \dots m_i$, it produces a stream of ciphertext blocks $c_1 c_2 \dots c_i$:

$$\begin{aligned} \text{PAD}_1 :=& \text{PRG} - sc(k, IV_1); & c_1 :=& \text{PAD}_1 \oplus m_1 \| IV_1 \\ \text{PAD}_2 :=& \text{PRG} - sc(k, IV_2); & c_2 :=& \text{PAD}_2 \oplus m_2 \| IV_2 \\ & & \dots & \\ \text{PAD}_i :=& \text{PRG} - sc(k, IV_i); & c_i :=& \text{PAD}_i \oplus m_i \| IV_i \end{aligned}$$

Use case and why it's called Unsynchronized mode: Alice/Bob encrypting/decrypting m_i/c_i can just do so by knowning k, IV_i, without any dependency with $j \neq i$ (e.g., no need to know IV_j or c_j). In other words, there is no state need to be maintained across blocks. It is called stateless mode of operation.

B. Block cipher mode of operation

Recall that Block cipher is a permutation selected by key k, whose input and output strings have the same length. In OTP +prf, it produces the ciphertext of double length with the message.

Block cipher mode of operation supports encryption with shorter ciphertext.

1) ECB: ECB is the most basic. $c := \langle \text{PRF}(m_1) \| \text{PRF}_k(m_2) \ldots \| \text{PRF}_k(m_l) \rangle$. ECB is deterministic.

2) CBC: **CBC** or cipher block chaining. Mode of encrypting a single variable-length message in CBC is: $c := IV \| \text{PRF}(IV \oplus m_1) \| \text{PRF}(c_1 \oplus m_2) \ldots \| \text{PRF}(c_{l-1} \oplus m_l)$

CBC is CPA-secure if PRF is pseudorandom function.

For encrypting multiple (variable-length) messages, CBC can use different IV for different messages. But there is another stateful approach, where the IV for the second message is (or reuses) the last ciphertext block in the first message. This stateful approach is called **chained CBC**. It is illustrated as in Figure 3.8 in the textbook.

Fact: chained CBC is not CPA-secure (one can construct a successful CPA-attack: $m_1^B = IV \oplus c_l^A \oplus m_1^A \Rightarrow c_1^B = c_1^A$ where m^A and m^B are two separate messages).

Summary: Modes of operation in encryption $\text{ENC}(m)$ are about encrypting multiple messages of variable length. Given a CPA-secure encryption is probabilistic, there are per-key randomness (as in s) and per-message randomness (as in IV). When encrypting a message of variable length, the randomness (s and IV) is fixed, and modes of operation deal with mapping variable-length messages to the block interface of block cipher. When encrypting multiple messages (under the same key), modes of operation deal with how to manage per-message randomness across different message? Two strategies: to reuse the randomness (stateful) or to simply get a fresh random IV for each message (stateless). For stateful, it can be reuse state of stream cipher, last ciphertext block, etc.

V. Message Authentication Code (MAC) 10/03/2019

Application scenario: Imagine Alice, a computer user, updates her OS by downloading software patches from Bob, the software-distribution server. In this scenario, Alice downloads software security patches in plaintext message m from Bob via the Internet. There is a malicious router $\mathcal{A}$ on the Internet which relays the message m from Bob to Alice. $\mathcal{A}$ can simply modify or forge a different software patch, say malware m', and relays that to Alice. Alice may end up running a malware m', instead of security patch m, on her computer.

To avoid this, question is how can Alice be rest-assured that the message Alice receives is identical to the message Bob sends? In other words, how can one detect the message-forging adversary $\mathcal{A}$?

A. The Scheme of MAC

Scheme: Π (GEN,MAC,VRFY)

$$\begin{aligned} t &\leftarrow \text{MAC}_k(m) \\ 1,0 &:= \text{VRFY}_k(m,t) \end{aligned}$$

Using the scheme in our scenario, Alice runs $t \leftarrow \text{MAC}_k(m)$ and sends out m, t to Bob. Adversary $\mathcal{A}$ may want to produce m', t' and fool Bob to run $\text{VRFY}_k(m', t') = 1$ successfully.

Correctness: Fixing k, $\forall m$, Π is correct iff $\text{VRFY}_k(m, \text{MAC}_k(m)) \equiv 1$.

B. Security definition

First, let's look at an example construction of MAC that, intuitively, is not secure. After that, we will describe how to formulate the security of MAC scheme.

In this example construction, we attempt to use the OTP-PRG encryption for providing authenticity. That is, given $c := \text{PRG}(k) \oplus m$, c is (ab)used as t. In $\text{VRFY}_k(m,t)$, it runs $\text{PRG}(k) \oplus t \stackrel{?}{=} m$. Clearly, this scheme is correct. But intuitively, it isn't secure: Flipping a bit in m and c/t results in a valid message,tag pair that passes VRFY. Also, the message-forging $\mathcal{A}$ can work out PRG (k) by $t \oplus m$, and then for any forged message m', forge $t' = (t \oplus m) \oplus m'$. This pair of m', t' will pass VRFY ().

Side-note: In this counter-example, it also says authenticity is not confidentiality. A EAV-secure encryption construction may not be a good candidate for MAC.

MAC Security: Given Π, $\mathcal{A}$, n, the adaptive chosen-message attack game $\text{MACFORGE}^{\text{ACMA}}$ is

$\text{MACFORGE}^{\text{ACMA}}_{\mathcal{A},\Pi}(n)$:

$\mathcal{C}$	$\mathcal{A}$
$k \leftarrow \text{GEN}(1^n)$	
	$m, t \leftarrow \mathcal{A}^{\mathcal{Q}(\text{MAC}_k(\cdot))}(1^n)$
$\text{VRFY}_k(m,t) \stackrel{?}{=} 1$	

Note that the oracle in this game $\mathcal{O}(\text{MAC}_k)$ is the following: Adversary $\mathcal{A}$ sends chosen messages $\{m_i\}$ to the oracle service $\mathcal{Q}(\text{MAC}_k)$ which hides the key k. Given $\{m_i\}$, $\mathcal{Q}(\text{MAC}_k(\cdot))$ returns tags $\{t_i\}$. Given $\{m_i, t_i\}$, $\mathcal{A}$ runs a PPT algorithm to produce m and to forge a tag t.

The success of the game is defined as $\text{MACFORGE}^{\text{ACMA}} = 1 \stackrel{\text{def}}{=} \text{VRFY}_k(m,t) = 1 \wedge m \notin \mathcal{Q}$.

A MAC scheme is **existentially unforgeable** under the ACMA game $\text{MACFORGE}^{\text{ACMA}}$ when

$$Pr[\text{MACFORGE}^{\text{ACMA}}_{\Pi,\mathcal{A}}(n) = 1] \leq \text{NEGL}(n) \tag{9}$$

Fact1: message authentication versus user authentication: message authentication ensures the message Bob receives is the same with what Alice sends. user authentication means Bob is convinced that the message received is indeed from Alice "the person".

Fact2: MAC versus error-correction code (ECC): 1) MAC detects while ECC both detects and recover. 2) MAC handles malicious message change while ECC handles random change of a small portion of the message. (Exercise XXX)

Fact3: timing attack on VRFY implementation. through side-channel timing info., one can gain advantage in guessing "correct" tag.

Fact4: MAC is not secure under replay attack.

C. Constructions

1) Fixed-length MAC *by* PRF*:*

$$\begin{aligned} \text{MAC}: &\quad t := \text{PRF}_k(m) \\ \text{VRFY}: &\quad t \stackrel{?}{=} \text{PRF}_k(m) \end{aligned}$$

Fact1: Canonical implementation of VRFY: When MAC is deterministic, one can reuse MAC for VRFY.

Security proof

To prove the MAC construction is ACMA-unforgeable, the key technique is to construct a **simulation** where the distinguisher $\mathcal{D}$'s view in PRF is simulated by $\mathcal{C}$ interacting $\mathcal{A}$ in $\text{MACFORGE}^{\text{ACMA}}$. The simulation is a type of reduction techniques used to proof security. Figure 4 illustrate the construction.

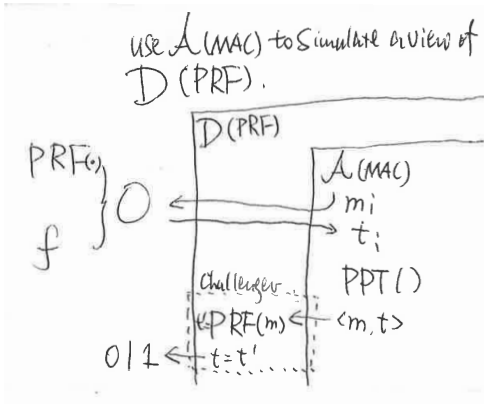

Fig. 4: Intuition of proving MAC security

Then, the general "template" for proving the security of a PRF-based construction is 1) proving the security (negligible probability of the game) given a truly random function, 2) use the simulation technique to prove the probability of the game given PRF is "indistinguishable" with that given truly random function (case 1).

2) Variable-length MAC *by* CBC-MAC *(§ 4.11):* Consider variable-length message $m = m_1, \ldots, m_l$ where each m_i is a block of length n. The CBC-MAC construction is:

$$\begin{aligned} \text{MAC}: \quad & t_0 := 0^n \\ & t_i := \text{PRF}_k(t_{i-1} \oplus m_i) \\ & t := t_l \\ \text{VRFY}: \quad & \|m\| = l \wedge t \stackrel{?}{=} \text{MAC}_k(m) \end{aligned} \tag{10}$$

Fact1: CBC-MAC versus CBC-mode encryption: The former uses zero IV (or t_0) while the latter uses random IV; the former outputs only t_l while the latter outputs all PRF results.

Fact2: arbitrary-length MAC is by using $m' = l, m_1, m_2, \ldots m_l$ as the input of fixed-length MAC.

D. Authenticated Encryption (10/08/2019)

1) Scheme:

$$\begin{aligned} c \quad &\leftarrow \quad \text{ENC}_k(m) \\ \{m, \perp\} \quad &:= \quad \text{DEC}_k(c) \end{aligned}$$

2) Security Definition:

$\text{AUTHK}^{\text{CPA}}(n)$:

$\mathcal{C}$	$\mathcal{A}$
$k \leftarrow \text{GEN}(1^n)$	
	$c \leftarrow \mathcal{A}^{\text{ENC}_k(\cdot)}(1^n)$
	$*\mathcal{Q} \stackrel{\text{def}}{=}$ All messages queried in $\mathcal{O}$
$m := \text{DEC}_k(c)$	

* *Footnote: We use red text to denote the part of security definition relevant to oracle.*

The success of the game is defined to be $\text{AUTHK}^{\text{CPA}}(n) = 1 \stackrel{\text{def}}{=} m \neq \perp \wedge m \notin \mathcal{Q}$.

The private-key encryption scheme Π is unforgeable when

$$Pr[\text{AUTHK}^{\text{CPA}}(n)] \leq \text{NEGL}(n) \tag{11}$$

3) Constructions: Construction: **enc-then-auth**. Given two keys k_e and k_m, the encryption algorithm is $\text{ENC}_{k_e}(m) \| \text{MAC}_{k_m}(c)$. Decryption algorithm is $\text{VRFY}_{k_m}(c, t) \stackrel{?}{=} 1?\text{DEC}_{k_e}(c) : \perp$.

Fact1: Conceptually, there are other two alternatives: auth-then-enc, and enc-and-auth. It can be easily seen that enc-and-auth is not secure as tag is revealed which leaks confidentiality.

Fact2: Two keys (k_m and k_e) must be independent! Counterexample: ENC is $\text{PRF}_k(m \| r)$ and MAC is $\text{PRF}_k^{-1}(c)$.

Fact3: Using authenticated encryption as it is without any state is vulnerable to **replay, reorder, reflection** attacks. Maintaining a message counter is the solution.

Fig. 5: Mallory attacking on message ordering by reordering and replay messages between Alice and Bob.

4) Beyond Data Integrity: Replay and reorder attacks: Consider Alice sends a sequence of messages to Bob, as in Figure 5. Bob wants the messages are received in the correct "order" as sent by Alice.

Even the messages are AE-encrypted, Mallory $\mathcal{A}$ can forge the order among messages and pass the verification. Given messages m_1, m_2, a reorder attacker forwards the messages in different order, say m_2, m_1. In applications of video streaming, reordering attacks may lead to Bob playing video frames out of order.

Given message m_1, a replay attacker forwards multiple copies of the message m_1, m_1 to Bob. In applications such as web shopping, replaying messages may lead to Bob (amazon.com) charges Alice's (amazon user) account twice.

AE does not defend replay or reorder attacks. Instead, maintaining a stateful monotonic counter synchronously on Alice and Bob will fix the problem. Alice will send out $\text{ENC}(m_1, mc_{Alice}++)$; to Bob who runs $\text{DEC}(c_1, mc_{Bob}++)$;.

VI. HASH FUNCTIONS (10/08/2019)

A. Scheme

Definition 6.1 (Fixed-length hash h): A fixed-length hash h is a compression function of its input space (called preimage space) larger than its output space (called digest space).

Definition 6.2 (Variable-length hash H): A variable-length hash H is a compression function: $H : \{0,1\}^* \to \{0,1\}^l$.

Formally, a variable-length hash function is a scheme of two algorithms, Π (GEN, H):

$$\begin{aligned} s &\leftarrow \text{GEN}(1^n) \\ y &:= H^s(x) \end{aligned}$$

Fact1: s is different from encryption key k in: 1) s is not private, 2) s is not chosen uniformly (and generated by GEN ()): There are "invalid" hash keys where $H^s(\cdot)$ is not legitimate.

B. Security

1) Collision Resistance: **Collision resistance**: collision exists (as long as $l' > l$), but it is difficult to find one. Formally, collision resistance is defined by HASHCOLL (n):

Security game HASHCOLL (n) is

$\mathcal{C}$	$\mathcal{A}$
$s \leftarrow \text{GEN}(1^n)$	
	$m_0, m_1 \leftarrow \mathcal{A}^{\mathcal{O}(H)}(1^n)$
$H(m_0) \stackrel{?}{=} H(m_1)$	

$$\text{HASHCOLL}(n) = 1 \stackrel{\text{def}}{=} H(m_0) = H(m_1) \wedge m_0 \neq m_1$$

H is a collision-resistant hash function (CR-HASH) iff. the following holds:

$$\forall \mathcal{A}, Pr[\text{HASHCOLL}(n) = 1] \leq \text{NEGL}(n)$$

2) Weaker Security: There are weaker security notions that collision resistance.

Second preimage collision resistance: informally, given the first preimage x, it is difficult to find the second preimage x', s.t., $H(x) = H(x')$.

Onewayness: informally, given the digest y, it is difficult to find a preimage x, s.t., $H(x) = y$.

C. Generic Attacks

1) Brute-force Attack: Instantiating the adversary in HASHCOLL gives us collision-finding attacks.

One attack strategy is to iterate through the space of digests 1^l (Recall $H : \{0,1\}^L \to \{0,1\}^l$). If we try $N = 2^l + 1$ distinct preimages, 100% chances are that there is at least one collision. This is due to pigeonhole principle (If we put $N + 1$ pigeons into N holes, at least one hole will end up with at least two pigeons.)

But if we want 50% chances to find a collision, how many samples of preimages does an attacker need to try? The answer is $\sqrt{N}$. It leads to birthday attacks.

2) Birthday Attack: In a Birthday attack, the adversary tries just $\sqrt{N} = 2^{\frac{L}{7}}2 + 1$ hash preimages, high chances (50%) are that she will find a collision among these preimages.

We can play a game in classroom: The teacher wants to find "Birthday collision" among students, that is, two students having the same day of their Birthdays (just the same day in a month). If the teacher samples 32 students, he will find one for certainty. But if he sample just $\sqrt{32} \approx 6$ students, high chances are he will find students having birthday collisions.

D. Hash-based Primitives

1) MAC domain-extension: Hash-and-MAC: A variable-length MAC, (GEN',MAC',VRFY') is constructed by a fixed-length MAC, (MAC,VRFY) and a variable-length hash function, (GEN,H). The construction is called **Hash-and-MAC**:

(12)

$$\begin{aligned} \text{GEN}'(1^n): &\quad k \leftarrow \{0,1\}^n, s \leftarrow \text{GEN}(1^n) \\ t := \text{MAC}'(m): &\quad \text{MAC}_k(H^s(m)) \\ \text{VRFY}'(t,m): &\quad \text{VRFY}(t, H^s(m)) \end{aligned}$$

Fact1: In Hash-and-MAC, the resultant MAC' is a secure variable-length MAC, when H is collision resistant and MAC' is an MAC.

Fact2: In hash-and-MAC, the "hash" is a var-length hash for domain extension, and the "MAC" is a fixed-length MAC.

2) Hash domain-extension (from h to H): Merkle-Damgard and Merkle trees: **Merkle-Damgard construction**: arbitrary-length hash (GEN,H) has arbitrary input length bounded by $L < 2^n$. Fixed-length hash (GEN,h) has input length $2n$ (w.l.o.g). Both have output length n. Merkle-Damgard construction constructs (GEN,H) by (GEN,H) as below:

- input $x := x_1, x_2 \ldots, x_B$, where each x_i is of block length n, and $B = L/n$.
-

$$\begin{aligned} H(x): \quad & y_0 := IV \\ & y_i := h^s(y_{i-1} \| x_i) \\ & y := y_B \end{aligned}$$

Fact1: CBC vs Merkle-Damgard: in CBC, an iteration is using PRF with $\oplus$, while in Merkle-Damgard, an interaction is h with concatenation.

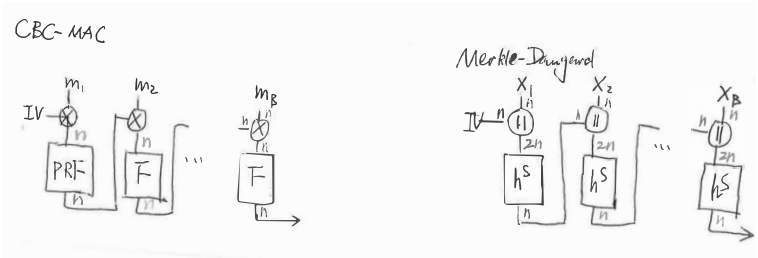

Fig. 6: Comparing Merkle-Damgard with CBC-MAC

Merkle hash tree or MHT is a construction paradigm that can be applied to different applications. One canonical application of Merkle tree paradigm, similar to Merkle-Damgard paradigm, is **hash domain extension**, constructing an arbitrary-length hash out of compression function. That is, $\text{GEN}_H, d \leftarrow \mathcal{MT}_t(b_1\|b_2\|...\|b_t)$, and the implicit verification is $\{1,0\} := \text{VRFY}(b_1\|...\|b_t, d)$ (Note that $b_1\|...\|b_t$ is a message of t blocks).

$$\mathcal{MT}_2(b_1, b_2) \stackrel{\text{def}}{=} h(b_1\|b_2)$$

Another application of Merkle tree paradigm is for **multi-message (set-membership) MAC** where the above scheme is extended with an interactive verification sub-protocol (i.e. query and verify of individual messages). Here, we only consider the first application of Merkle tree (i.e. hash domain-extension) to describe its security.

$$\mathcal{MT}_2(m_1, m_2) \stackrel{\text{def}}{=} h(h(m_1)\|h(m_2))$$

The security definition (collision resistant) of Merkle tree is formulated in similar way to that of hash (collision-finding game).

Theorem 6.3: The security of Merkle tree relies on the security of underlying hash. Formally, if (GEN_H, H) is collision resistant and t is fixed, $(\text{GEN}_H, \mathcal{MT}_t)$ is collision resistant.

Proof We prove by contradiction. If $\mathcal{MT}_H$ is not collision resistant, H_H is not collision resistant. W.l.o.g., we consider $t = 2$.

$\mathcal{MT}_H(\cdot)$ is not collision resistant, thus a PPT $\mathcal{A}$ can easily find $m_1^a\|m_2^a, m_1^b\|m_2^b$ s.t. $\mathcal{MT}(m_1^a\|m_2^a) = \mathcal{MT}(m_1^b\|m_2^b)$. $\mathcal{MT}(m_1^a\|m_2^a) = H(H(m_1^a)\|H(m_2^a))$.

$\therefore H(H(m_1^a)\|H(m_2^a)) = H(H(m_1^b)\|H(m_2^b))$

$\therefore H(m_1^a)\|H(m_2^a) = H(m_1^b)\|H(m_2^b)$. (Otherwise, we find a collision for $H(\cdot)$).

$\therefore H(m_1^a) = H(m_1^b)$. $\therefore m_1^a = m_1^b$. (Otherwise, we find a collision for $H(\cdot)$)

Similarly, $m_2^a = m_2^b$.

Therefore, $m_1^a m_2^a = m_1^b m_2^b$ which contradicts the assumption $m_1^a m_2^a \neq m_1^b m_2^b$.

This proves that for $\mathcal{MT}_2$, the equality of the parent node implies the equality of the two children nodes. More generally, with $t = 2^x > 2$, one can apply the proved *recursively*. Thus, the equality of the root node implies the equality of all t children nodes, hence $m_1\|...\|m_t$, contradiction.

Theorem 6.4: Merkle tree of variable-length t is not collision resistant. Specifically, one can find two sets of inputs $m_1\|...\|m_t$ and $m'_1\|...\|m'_{2t}$, such that $MT_t(m_1\|...\|m_t) = MT_{2t}(m'_1\|...\|m'_{2t})$.

Proof $\forall z = H(x\|y)$, $\mathcal{MT}_2(z\|z) = H(z\|z) = H(H(x\|y)\|H(x\|y)) = \mathcal{MT}_4(x\|y\|x\|y)$

E. Hash Applications

Fingerprinting or hash pointer: using $H(\cdot)$ to digest message m, where m can be virus ID (for virus detection), cloud file (for de-duplicated cloud storage), P2P file (for P2P file lookup)

Merkle tree: $\mathcal{MT}(m_1, m_2..., m_t)$ as domain-extension of H for hashing multiple messages. The application is to digest multiple files while allowing client to access individual files in cloud storage.

Using hash for fingerprinting and Merkle tree requires the collision resistance of hash functions.

key derivation and password hashing: key derivation is a mapping of non-uniform, high min-entropy distribution to a uniform distribution. One application is to derive (uniform) key, required by private-key encryption, from non-uniform strings, such as passwords, bio-metric data, etc.

Using hash for key derivation requires the random-oracle model of hash functions.

VII. KEY EXCHANGE (§ 10)

A problem left pending in private-key schemes is how Alice and Bob can establish the private key initially? The first baseline is that Alice directly sends the generated secret key to Bob. However, this may disclose the secret key to Eve (over the insecure channel). The second baseline is that Alice distributes the secret key to Bob over an encrypted channel; say with the key encrypted (but what is the key used for encrypting the key?).

Secure key distribution over an insecure channel is handled by key-exchange protocols. Conceptually, key distribution over a secure channel (e.g., meet in person) is trivial and has limited applications (only in military settings).

We will talk about two constructions of key exchange; a two-party protocol, DHKE (§ VII-B) and a three-party protocol, KDC and Needham-Schroeder (§ VII-C). The latter handles key exchange among multiple parties in an open system with transient interactions.

A. Definitions: Schemes and Security

KE Scheme (Protocol): A KE scheme is an interactive protocol: Given Alice and Bob, a protocol Π consists of a set of instructions and is executed by A and B, resulting in a transcript of messages exchanged between them. The following describes the KE-protocol execution.

1 Initially, A and B hold 1^n.
2 Π is interactively executed between A and B, with messages exchanged in multiple rounds. The trace of messages being exchanged is called transcript.
3 Finally, the protocol execution produces key k_A on A and k_B on B.

Protocol correctness is defined as $k_A \stackrel{\text{def}}{=} k_B$.

Protocol security is defined in the security game $\text{KE}^{\text{EAV}}_{\mathcal{A},\Pi}$ below. This game runs between challenger $\mathcal{C}$ and an eavesdropping $\mathcal{A}$. $\mathcal{C}$ models the protocol execution between two running parties, $\mathcal{C}_A, \mathcal{C}_B$.

$\text{KE}^{\text{EAV}}_{\mathcal{A},\Pi}$:

$\mathcal{C}$	$\mathcal{A}$
$\mathcal{C}_A, \mathcal{C}_B$ joinly run protocol Π, producing transcript TRANS and $k \stackrel{\text{def}}{=} k_A = k_B$	
Random bit $b \leftarrow \{0,1\}$, Another "key" $\begin{cases} k' = k & \text{, if } b = 0 \\ k' \leftarrow 1^n & \text{, if } b = 1 \end{cases}$	
	$b' \leftarrow \mathcal{A}^{PPT}(\text{TRANS}, k')$

Success of game $\text{KE}^{\text{EAV}}_{\mathcal{A},\Pi} \stackrel{\text{def}}{=} b = b'$.
Security of a KE protocol is defined as $Pr[\text{KE}^{\text{EAV}}_{\mathcal{A},\Pi}] - \frac{1}{2} < \text{NEGL}(n)$

Example: The baseline key-exchange by sending out secret key in the clear is unsecured. In this baseline construction, transcript TRANSis key k. Adversary can then easily have non neglibability adversarial advantage that breaks the KE security definition.

B. Construction (DHKE) and Security Analysis

1) Background: Group theory (Chapter 8 in textbook): A **group** defines a set of numbers and *an* operation, associated with a bunch of rules. The set of numbers are commonly $\{0, 1, \ldots, N-1\}$ modulo N. The operation can be addition or multiplication. There are common rules on the operation such as closure, identity, invertible, associativity. There are derived rules: commutativity and cancellation law.

$\mathbb{Z}^*{}_N$ denotes a multiplicative group of integers modulo N. N is group modulo base. The group contain number 1 as the identity element and the numbers that are relatively prime to N. A multiplicative group does not contain number 0. For instance, $\mathbb{Z}^*{}_{15} = \{1, 2, 4, 7, 8, 11, 13, 14\}$.

Group order q is the number of elements in the group, $|\mathbb{Z}^*{}_N|$. For instance, q of $\mathbb{Z}^*{}_{15}$ is 8. In general, $q = \Phi(N)$. $\forall e \in \mathbb{Z}^*, e^q = 1 \mod N$.

Group generator: Each group element $\forall e \in \mathbb{Z}^*$ has a per-element order $q(e)$ which is the smallest value s.t. $e^{q(e)} = 1 \mod N$. Each group element g can generate a subgroup of $\mathbb{Z}^*$, by $\{g^0, g^1, g^2, \ldots\}$. The order of an element is the order of the subgroup generated by the element. And g is the generator of the subgroup.

2) Group-theoretic Problems and Hardness: **Discrete logarithm**: Discrete log is defined by $\forall h \in \mathbb{Z}^*$, there is a unique $x \in \mathbb{Z}^*{}_q$, s.t. $h = g^x$. x is the discrete log of h.

A DL problem, given $\mathbb{Z}^*{}_q$ with generator g, is to compute $x = \log_q h$ for a uniform element $h \in \mathbb{Z}^*$.

Fact: A DL problem is a logarithm operation with elements defined in the domain of groups.

DH (Diffie-Hellman) problems: Computational DH, $\text{CDH}(h_1, h_2, h_3)$: Given $h_1 = g^x$ and $h_2 = g^y$, it is hard to compute $h_3 = g^{xy}$. Given group exponentiation is easy, the hardness of CDHrequires the hardness of DL (if DL is easy, one can obtain x from g^x and compute $(g^y)^x$).

Decisional DH, DDH: Given $h_3 = \text{CDH}(h_1, h_2)$, it is hard to decide if it is different from a uniform group element. DDHis used to prove the security of DH protocol.

3) DHKE: **Execution between Alice and Bob**:

Alice		Bob	(13)
$(\mathbb{G}, q, g) \leftarrow \mathcal{G}$			
$x \in \mathbb{Z}_q$			
	$\xrightarrow{g^x}$		
		$y \in \mathbb{Z}_q$	
	$\xleftarrow{g^y}$		
$k_A = (g^y)^x$		$k_B = (g^x)^y$	

1 Alice runs PPT algorithm: $(\mathbb{G},q,g) \leftarrow \mathcal{G}$
2 Alice chooses uniform $x \in \mathbb{Z}_q$ and computes $h_A := g^x$.
3 Alice sends $(\mathbb{G}, q, g, h_A)$ to Bob
4 Bob receives $(\mathbb{G}, q, g, h_A)$. He chooses $y \in \mathbb{Z}_q$ and computes $h_B := g^y$. Bob sends h_B to Alice and outputs key $k_B := h_A^y$
5 Alice receives h_B and outputs key $k_A := h_B^x$.

Correctness: $k_A := h_B^x = (g^y)^x = (g^x)^y = h_A^y = k_B$.

Security: Proof of security depends on the hardness of decisional DH problem (DDH). DDHfurther relies on the hardness of DL problem (given h and y find x s.t. $h^x = y$).

Concretely, suppose a DL problem involving a cyclic group $\mathbb{Z}^*{}_N = \{g^0, g^1, \ldots, g^{q-1}\}$, where q is group order, g is a generator and N is modulo (which may be prime number). DH protocol works efficiently because group exponentiation is easy. DH protocol is secure because $\text{CDH}(h_A, h_B, k)$ is hard.

Fact1: basic DH construction is not secure against active adversary, including impersonation attacker who plays the role of Alice to Bob, and man-in-the-middle attacker who modifies the messages exchanged between Alice and Bob.

C. Key-Distribution Center (KDC) for 3-Party Key Exchange

We consider extending the two-party DHKE into three-party KE. That is, given shared key k_A between A and KDC (as B) and k_C between KDC and C, how can A and C exchange keys? This is useful in real applications in P2P networks.

Needham-Schroeder protocol based on KDC: a key-distribution protocol using KDC in the Kerberos authentication service.

0 Alice joins the network: running key exchange to share k_A between Alice and KDC.
1 Bob joins the network: running key exchange to share k_B between Bob and KDC.
2 Alice wants to communicate with Bob: running Needham-Schroeder to share session key between them.
    1 Alice to KDC: "Alice wants to talk with Bob"
    2 KDC: $k \leftarrow \{0,1\}^n$
    3 KDC to Alice: $\text{ENC}_{k_A}(k), \text{ENC}_{k_B}(k)$
    4 Alice to Bob: "Let's talk, $\text{ENC}_{k_B}(k)$". $\text{ENC}_{k_B}(k)$ sent by Alice is called ticket in the protocol of Needham-Schroeder and Kerberos. It is an encrypted session key and serves as a credential.

VIII. Public-Key Encryption

A. Introduction: Public-Key Cryptography

Suppose a website conducts a survey among its users by collecting private user information. Assume there are N users. For private-key encryption schemes, it requires setting up N secret keys. Using public key cryptography, it allows the website posts one public key and all N users using the public key to send encrypted private information.

The benefits brought by PKC is 1) simplified (public) key distribution, 2) reduced number of secret keys the server needs to maintain (one secret key in PKC versus N secret keys in SKC).

We will introduce public-key encryption (PKE) here and digital signature (DS) in § IX.

B. Schemes

Scheme: similar to PRIVKE scheme, except that $pk, sk \leftarrow \text{GEN}(1^n)$.

PKEscheme $\Pi^{\text{PKE}}(\text{GEN}, \text{ENC}, \text{DEC})$ consists of three PPT algorithms:

$$\begin{aligned} pk, sk &\leftarrow \text{GEN}(1^n) \\ c &\leftarrow \text{ENC}_{pk}(m) \\ m &:= \text{DEC}_{sk}(c) \end{aligned}$$

Execution between Sender and Receiver

Sender		Receiver
		$pk, sk \leftarrow \text{GEN}(1^n)$
	$\xleftarrow{pk}$	
$c \leftarrow \text{ENC}_{pk}(m)$		
	$\xrightarrow{c}$	
		$m' := \text{DEC}_{sk}(c)$

(14)

C. Security Definition

PKE schemes is different from those of PrivKE in that its EAVgame is defined the same to CPAgame.

Security definition(IND-EAV):

Game $\text{PRIVK}^{\text{EAV}}(n)$:

$\mathcal{C}$	$\mathcal{A}$
	$m_0, m_1 \leftarrow \mathcal{A}(pk)$
$b \leftarrow \{0,1\}$	
$c \leftarrow \text{ENC}(pk, m_b)$	
	$b' \leftarrow \mathcal{A}(pk, c)$

Successful game: $b = b'$, denoted by $\text{PRIVK}^{\text{EAV}}(n)$.
Security definition: $Pr[\text{PRIVK}(n) = 1] - \frac{1}{2} \leq \text{NEGL}(n)$.

Game $\text{PRIVK}^{\text{CPA}}(n)$:

$\mathcal{C}$	$\mathcal{A}$
	$\{m_0^i, m_1^i \vert \forall i\} \leftarrow \mathcal{A}(pk)$
$b \leftarrow \{0,1\}$	
$c \leftarrow \text{ENC}(pk, \{m_b^i \vert \forall i\})$	
	$b' \leftarrow \mathcal{A}^{\mathcal{Q}()}(pk, c)$

$\mathcal{Q}$ is defined by adversary can query $c_b \leftarrow \mathcal{Q}(m_0, m_1)$, but without knowing b. In other words, b is hidden in the public-key encryption oracle.

D. Domain extension

The ciphertext of multiple message blocks is the concatenation of ciphertext blocks, each produced by encrypting one message block.

$\text{ENC}'_{pk}(m_1 \| m_2 \| ... m_l) = \text{ENC}_{pk}(m_1) \| \text{ENC}_{pk}(m_2) \| ... \text{ENC}_{pk}(m_l)$

E. Construction: El Gamal

$$
\begin{aligned}
pk, sk \leftarrow \text{GEN}(1^n): \quad & \mathbb{G}, q, g \leftarrow \mathcal{G}(n) \\
& x \leftarrow \mathbb{Z}_q (\stackrel{\text{def}}{=} \{1, g^1, g^2, ...g^{q-1}\}) \\
& sk = x, \mathbb{G}, q, g \\
& pk = h (\stackrel{\text{def}}{=} g^x), \mathbb{G}, q, g \\
c \leftarrow \text{ENC}_{pk}(m): \quad & pk = h, \mathbb{G}, q, g \\
& y \leftarrow \mathbb{Z}_q \\
& c = g^y (\stackrel{\text{def}}{=} c_1), h^y \cdot m (\stackrel{\text{def}}{=} c_2) \\
m' := \text{DEC}_{sk}(c): \quad & m' = c_2 / {c_1}^x
\end{aligned} \tag{15}
$$

Note that $\mathbb{G}$ and (g,q) are two groups. The second group $\mathbb{Z}_q \stackrel{\text{def}}{=} \{1, g, g^2, ...g^{q-1}\}$. The first group $\mathbb{G}$ is a subgroup of $\mathbb{Z}_q$.

Correctness: $m' = c_2/{c_1}^x = h^y \cdot m/(g^y)^x = (g^x)^y \cdot m/(g^y)^x = m$.

Security (informal and intuitive): transcript $pk = h, \mathbb{G}, q, g$ and $c_1 = g^y, c_2 = h^y \cdot m$ is disclosed to Eve. Informally, discrete log problem prevents Eve from knowing x, y from $h = g^x, c_1 = g^y$. (Otherwise, knowing x, y, Eve can infer g^x).

Example 11.17: Let q =83 and $p = 2q+1 = 167$. Let $\mathbb{Z}_q = \{1, 2, 3, 4, ...82\}$. Let $\mathbb{G}$ denote the group $\{h^2 \mod p | \forall h \in \mathbb{Z}_q\}$; this group is called quadratic residues module p. $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p$.

Take $g = 2^2 = 4 \mod 167$ to be the generator. Say the secret key $37 \in \mathbb{Z}_{83}$. The public key is $pk = p, q, g, h = 167, 83, 4, 4^{37} \mod 167 = 167, 83, 4, 76$[2]

We use p to represent $\mathbb{G}$ and g, q to represent $\mathbb{Z}_q$.

Say a sender encrypts message $m = 65 \in \mathbb{G}$ (note $65 = 30^2 \mod 167$). If $y = 71$, ciphertext is $4^{71} \mod 167, 76^{71} \cdot 65 \mod 167 = 132, 44$.

IX. Digital Signatures (§ 12)

A. Scheme

Scheme:

SIGscheme $\Pi^{\text{SIG}}(\text{GEN}, \text{SIGN}, \text{VRFY})$ consists of three PPT algorithms:

$$
\begin{aligned}
pk, sk &\leftarrow \text{GEN}(1^n) \\
\delta &\leftarrow \text{SIGN}_{sk}(m) \\
\{1, 0\} &:= \text{VRFY}_{pk}(m, \delta)
\end{aligned}
$$

Correctness is defined as $\forall m, \{pk, sk\}, \text{VRFY}_{pk}(m, \text{SIGN}_{sk}(m)) \equiv 1$.

Execution between Sender and Receiver:

Sender		Receiver
$pk, sk \leftarrow \text{GEN}(1^n)$		
	$\stackrel{pk}{\rightarrow}$	
$\delta \leftarrow \text{SIGN}_{sk}(m)$		
	$\stackrel{m, \delta}{\rightarrow}$	
		$b := \text{VRFY}_{pk}(m, \delta)$

(16)

Compare and contrast: Compare the execution of digital signature in Equation 16 with that of public-key encryption in Equation **??**. We can see that GEN in digital signature runs on the sender while GEN in public-key encryption runs on receiver. In digital signature, it is like the sender pushes the message to the receiver, while in public-key encryption, it is more like the receiver pulls the message from the sender.

[2]One can write an efficient algorithm to do group exponentiation and calculate $4 * 37 \mod 167 = 76$.

B. Benefits and Distinction of Digital Signatures

1. Easy key distribution and management: Distribute public key instead of private key.

In addition, the number of secret keys a sender maintains in digital signatures is one. Whereas, the number of secret keys in MAC is N, where N is the number of receivers. The saving in the number of secret keys is significant, for applications involving a large number of receivers (think about distributing Windows OS patches to millions of Windows users as an example).

2. Public verifiability: Note that in digital signatures, receiver does not need any secret key to verify the integrity of message. Using a public key suffices to verify the message authenticity. This property is called public verifiability. It implies *transferability*, that is, one receiver who verifies can simply passes public information (i.e., $\{m, \delta\}$) to the next "receiver" to verify. There is no need of setting up secret keys in transferring verifiable information, which is unlike MAC.

3. Non-repudiation: The digital signature of message m can be used as an evidence that the sender has signed message m in the past. In other words, with digital signature, the sender cannot deny the fact that he has signed the message. Because the sender is the only one who knows the secret/signing key, the digital signature presents itself as a proof of the sender's signing action in the past.

By contrast, a MAC tag does not provide non-repudiation and cannot be used as an evidence. A counterexample is the following: Consider the receiver uses a MAC tag (i.e., t =MAC (m))) as an evidence that the sender has authenticated message m before. Such evidence is not trustworthy (and a judge cannot take it), because a malicious receiver, knowing the secret/authentication key, can simply produce t on a message that the sender has not authenticated before.

C. Security Definition

Game $\text{SIG-FORGE}_{\mathcal{A},\Pi}(n)$:

$\mathcal{C}$	$\mathcal{A}$
	$m, \delta \leftarrow \mathcal{A}^{\mathcal{O}(\text{SIGN}_{sk}(\cdot))}(pk) \stackrel{\text{def}}{=} \mathcal{A}(pk, \mathcal{Q}*)$
$b := \text{VRFY}_{pk}(m, \delta)$	

*: $\mathcal{Q}$ is defined as the set of message-signature pairs $\{m', \delta'\}$ where $\mathcal{A}$ sends each m' to the signing oracle and receives signature δ'. $\mathcal{Q}$ covers the case that adversary observes the transcript between sender and receiver (that is, m,δas in the "execution of digital signature").

Successful game: $\text{SIG-FORGE}(n) = 1 \stackrel{\text{def}}{=} b = 1 \wedge m \notin \mathcal{Q}$.

Security definition: $Pr[\text{SIG-FORGE}(n) = 1] \leq \text{NEGL}(n)$.

D. Construction: Domain Extension

Domain extension is to support the message of variable length.

Hash-and-sign paradigm for domain extension in digital signatures.

$pk, sk \leftarrow \text{GEN}'(1^n)$:	$pk, sk \leftarrow \text{GEN}(1^n)$
	$s \leftarrow \text{GEN}_H(1^n)$
$\delta \leftarrow \text{SIGN}'_{sk}(m)$:	$\delta \leftarrow \text{SIGN}_{sk}(H^s(m))$
$b := \text{VRFY}'_{pk}(m, \delta)$:	$\text{VRFY}_{pk}(H^s(m), \delta)$

If the hash is collision resistant and fixed-length SIGis unforgeable, the variable-length SIGconstructed by hash-and-sign is unforgeable.

Counterexample: If the hash is not collision resistant, construct an attack to break the digital signature (or the security definition in SIG-FORGE): Adversary observing $\{m,\delta\}$ will find hash collision with m easily. Name the colliding message by m', such that $H^s(m) = H^s(m')$. Adversary can present $\{m', \delta\}$ as the forged message to the receiver. Verification will pass because $\text{VRFY}'(m', \delta) = \text{VRFY}(H^s(m'), \delta) = \text{VRFY}(H^s(m), \delta) = 1$.

E. Construction (Fix-length): Schnorr Identification

Overall, a digital signature scheme can be constructed from an interactive identification protocol (through Fiat-Shamir transform). We talk about the scheme definition of identification protocol (§ IX-E2), Fiat-Shamir transform (§ IX-E1), and a DL based construction of identification (§ IX-E3).

1) Identification scheme: Definition: **Protocol execution** between Prover (of her secret key sk) and Verifier (of Prover's public key pk.

$$
\begin{array}{rcl}
\text{Prover}(sk) & & \text{Verifier}(pk) \\
(I, st) \leftarrow P_1(sk) & & \\
& \xrightarrow{\text{Initial msg } I} & \\
& & r \leftarrow \Omega_{pk} \\
& \xleftarrow{\text{Challenge } r} & \\
s := P(sk, st, r) & & \\
& \xrightarrow{\text{Response } s} & \\
& & V(pk, r, s) \stackrel{?}{=} I
\end{array}
\tag{17}
$$

The security of identification scheme is, informally, that given the transcript (I, r, s), the adversary without knowing secret key sk cannot generate s and use it to fool verifier into accepting.

2) Fiat-Shamir transform:

$$
\begin{array}{rl}
\text{GEN}'(1^n): & sk, pk \leftarrow \text{GEN}_{ID}(1^n) \\
\delta := \text{SIGN}_{sk}(m): & I, st \leftarrow P_1(sk) \\
& r := H(I, m) \\
& s := P_2(sk, st, r) \\
& \delta = (r, s) \\
\text{VRFY}_{pk}(m, \delta): & V(pk, r, s) \stackrel{?}{=} I \\
& H(I, m) \stackrel{?}{=} r
\end{array}
$$

See Figure 7 about the transform.

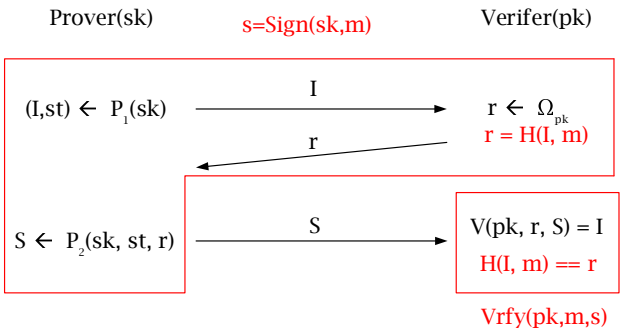

Fig. 7: Intuition of Fiat-Shamir transform

Fact: ID protocol is simpler, interactive; and digital signature is extensible (w.r.t. message) and non-interactive.

3) Construction: Schnorr Identification: **Schnorr identification protocol execution**: Secret key is $x \in \mathbb{Z}_q$ and public key is $\mathbb{G}, g, q, y$. Here $y = g^x$.

$$
\begin{array}{rcl}
\text{Prover}(sk) & & \text{Verifier}(pk) \\
k \leftarrow \mathbb{Z}_q; I = g^k & & \\
& \xleftarrow{I} & \\
& & r \leftarrow \mathbb{Z}_q \\
& \xrightarrow{r} & \\
s = [rx + k] \mod q & & \\
& \xrightarrow{s} & \\
& & \text{Check } g^s \cdot y^{-r} \stackrel{?}{=} I
\end{array} \tag{18}
$$

Correctness: $g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^k = I$

Security proof: By security definition in § IX-E1, the adversary can do the following: Given public key pk, she can send message I, receive r and send s such that $g^s \cdot y^{-r} == I$.

If this is the case, then the adversary can do it twice with the same initial message I. She can obtain two transcripts like this: I, r_1, s_1 and I, r_2, s_2, such that $g_1^s \cdot y^{-r_1} == I$ and $g_2^s \cdot y^{-r_2} == I$. Thus,

$$g^{s_1 - s_2/(r_1 - r_2)} = y$$

One can simply calculate $\log_g y$ to be $\frac{s_1 - s_2}{r_1 - r_2} \mod q$ which contradict the hardness assumption on DL .

F. Challenge-response protocols for user authentication (Chapter 6 in Book [2])

In this subsection, we take a different perspective: Instead of constructing a digital signature scheme based on an identification protocol, we construct an identification/user authentication protocol from public-key cryptographic primitives including digital signatures. The intuition is illustrated in Figure 8.

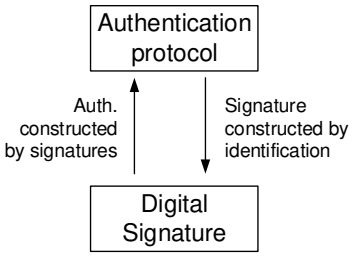

Fig. 8: Two perspectives: 1) Authentication protocol constructed by digital signature as primitive, and 2) digital signature constructed by identification protocols (as in Fiat-Shamir transform)

1) Case: Two-factor authentication: In this game, the authentication server sends a passcode to a user's registered phone, as a challenge to the user. The user responds by sending the passcode to the server, as a proof that she has access to the registered phone.

In general, authentication server sends a challenge and user sends a response. The server verifies the user response (as a proof) against the challenge.

The response needs to be 1) something that only a legal user can generate and 2) also has to do with the challenge.

2) Case: Public-key based authentication: Public-key based challenge-response protocols are described in the identification protocol (§ IX-E1).

Baseline design 1: To construct it, one baseline is using digital signature. Consider authentication server Bob and user Alice. In the first round, the challenge is a random number r picked by Bob. In the second round, the response is a message signed by Alice's secret key: $\text{SIGN}_{sk}(\text{“}Alice'', r)$. The server Bob verifies by verifying the signature.

However, this design is not secure, if the man-in-the-middle attacker changes r and Alice, the legitimate user, end up with signing r' and get declined by the server.

Baseline design 2: Another baseline construction is using public key encryption. In the first round (for challenge), the challenge is Bob sends a random number r encrypted by Alice's public key, that is, $\text{ENC}_{pk}(r)$. In the second round, the response is the plaintext of r sent by Alice. Because only Alice can decrypt the ciphertext challenge. The server simply checks the equality.

This baseline is not secure, as the adversary can simply changes the plaintext response $s = r$ to any other value, failing legal user Alice's authentication.

The secure construction: It uses both public-key encryption and digital signature.

Prover(sk)		Verifier(pk)
		$r \leftarrow \text{ENC}_{pk_A}(\text{SIGN}_{sk_B}(\text{“A”},R))$
	$\xleftarrow{r}$	
$s \leftarrow \text{ENC}_{pk_B}(\text{SIGN}_{sk_A}(\text{“A”},\text{“B”},R))$		
	$\xrightarrow{s}$	

(19)

X. TLS (§ 12.7/8)

A. Overview: TLS-HS and TLS-RL

Target scenario: A web browser wants to communicate securely with a web server (identified by its IP address). Using HTTPS, the network traffic is encrypted by private-key cryptography schemes. TLS or transportation layer security is the protocol underneath HTTPS: It consists of two phases: an initial hand-shake process (TLS-HS) that establishes shared secret keys between browser and web server, and a record-layer process (TLS-RL) that uses shared keys to authenticated-encrypt the messages issued by the applications/users. There are four secret shared keys, each two for communication in a direction. Different keys are also used for message authentication and encryption, separately. For simplicity/educational purpose, we consider TLS-HS establishes one shared key between browser and web server.

B. TLS-HS Design Iteration 1: DHKE under MitM Attacks

Our focus is to design TLS-HS protocol from scratch. We will go over several design iterations to refine the protocol.

To start with, the first design is use DHKE for TLS-HS. As we know DHKE provides the confidentiality of shared key against passive adversary (i.e., an eavesdropper who passively observes the network traffic without changing it). However, in practice, there are active adversaries who can forge the network traffic.

MITM attacks: Man-in-the-middle attack (MITM) is an “active” attack where the adversary actively modifies the messages transmitted in the execution of a KEYEXCH protocol. Concretely, Adversary $\mathcal{A}$ intercepts the first-round DH message g^x from Alice A and replaces it by g^{xx} before sending it to Bob B; g is group generator and xx, yy are generated by $\mathcal{A}$. She also does the same in the second round by replacing Bob B's message (g^y) with g^{yy}. The protocol execution is illustrated as below.

$$A \xrightarrow{g^x} \mathcal{A} \xrightarrow{g^{xx}} B \tag{20}$$

$$A \xleftarrow{g^{yy}} \mathcal{A} \xleftarrow{g^y} B \tag{21}$$

With this attack, $\mathcal{A}$ can forge a shared key $g^{x \cdot yy}$ with Alice A and a different shared key $g^{xx \cdot y}$ with Bob B.

C. TLS-HS Design Iteration 2: DHKE + Digital Signatures

TLS-HS defends the MITM attack by authenticating the transcript of the protocol execution. Concretely, A will check whether the transcript she receives is identical to the authentic transcript B receives. B's transcript is signed by B's (or server S) private key, which is certified by $\mathcal{CA}$. As $\mathcal{A}$ can not forge a $\mathcal{CA}$'s certification, MITM will not succeed. See Equation 22 for details (in red is what TLS-HS adds to DHKE).

Alice		Bob
$(\mathbb{G}, q, g) \leftarrow \mathcal{G}, pk_A, sk_A$		pk_B, sk_B
$x \in \mathbb{Z}_q, \delta_A = \text{SIGN}_{sk_A}([g^x, pk_A])$		
	$\xrightarrow{g^x, \delta_A}$	
		$\text{VRFY}_{pk_A}([g^x, pk_A], \delta_A)$
		$y \in \mathbb{Z}_q, \delta_B = \text{SIGN}_{sk_B}([g^y, pk_B])$
	$\xleftarrow{g^y, \delta_B}$	
$\text{VRFY}_{pk_B}([g^x, pk_B], \delta_B)$		
$k_A = (g^y)^x$		$k_B = (g^x)^y$

(22)

TLS-HS is secure under MITM attack, as the adversary cannot forge the signature using A or B's secret key. In the above figure, $\mathcal{A}$ cannot forge $\text{SIGN}_{sk_A}(\text{“}g^x, pk_A\text{”})$ without secret key sk_A.

D. TLS-HS Design Iteration 3: PKI and CA

In TLS-HS, the underlying assumption is that A and B (or web browser and server) are identified by their public key. This causes problems in practice: Consider a web browser connected to a web server for the first time. This web browser can only receive the public key of the server from the server itself (assuming no other online trusted party). When the web server is malicious (attempting to impersonate another domain), the web server can simply generate a public key and claims it is bound to an impersonated domain. In other words, how can a web client establish trust about the server's identity? Or how can a client trust the binding between a public key and a web server.

The solution is Public Key Infrastructure (or PKI). In PKI, trusted third party, called certificate authority (CA), exists to issue certificates on the binding between the domain name of a web server and its pubic key. Formally, a **certificate**, $\text{CERT}_{\mathcal{CA}} \rightarrow S$, is a signed statement that binds the identity of server S to its public key pk_S. The certificate is signed by $\mathcal{CA}$'s private key and verified by its public key. $\mathcal{CA}$, the certificate authority, is fully trusted to the extent that whatever statement it issues is trusted. A certificate is below:

$\text{CERT}_{\mathcal{CA}\rightarrow S} \stackrel{\text{def}}{=} \text{SIGN}_{sk_{\mathcal{CA}}}(binding(\text{“}S \text{ identity”}, \text{“}S \text{ public-key”}\ pk_S)$

Fact: A certificate is like a ID card, where the ID card issued by the government ($\mathcal{CA}$) certifies the binding between a subject's appearance (server S's public key) and his identity (S's identity).

Summary: Overall, bootstrapping trust for secure communications is realized by running three protocols among three parties: certificate authority $\mathcal{CA}$, server S, and client C. The whole process runs by three protocols at three stages: 1) At the very beginning, it runs the general PKI protocol as three-party protocol (among S,$\mathcal{CA}$,C) to securely set up environments (distribute certificates and public keys) for the next stage, 2) At the second stage, it runs the TLS-HS protocol as two party protocol between S and C to establish the trust and shared keys between them, 3) At the third stage, it runs the TLS-RL (record layer) protocol between S and C to securely communicate between them using the shared secret keys. In particular, the record-layer protocol runs private-key authenticated encryption, with sequence numbers and the four private keys (generated in TLS-HS and used pairwise for encryption and authentication).

E. PKI in practice (Nov. 12)

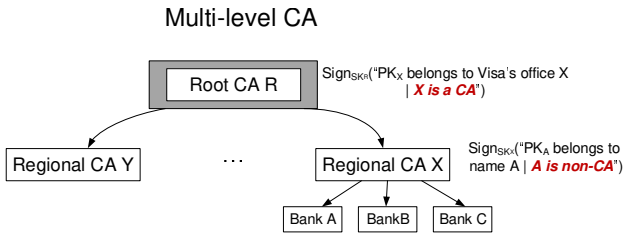

Fig. 9: Multi-level CA

Multi-level CA: Practical CAs are hierarchical: there is a root-level CA and multiple regional CAs. Consider Visa as an example, illustrated as in Figure 9: 1) There are regional offices in Visa. The Visa office in CNY authenticates Niagara Bank. This is done by issuing certificate like [$pk_{\text{NiagaraBank}}$ belongs to name 'Niagara Bank']; The certificate is signed by regional CA's secret key $sk_{\text{Visa\_CNY}}$. 2) The job of the root CA is to certificate the authority of regional CAs. This is done by issuing certificate like [$pk_{\text{Visa\_CNY}}$ belongs to regional CA named by 'Visa_CNY']; The certificate is signed by root CA's secret key $sk_{\text{Visa\_root}}$.

There are two kinds of "authorities", CAs and non-CAs (like the bank). A classic attack in multi-level CA is to (mis)use a non-CA's secret key to certify another fake non-CA. For instance, in Microsoft's IE under PKI, a misusing domain owner signs a certificate of a normal domain name with a fake public key. The security fix for this is to distinguish whether the public key in a certificate is CA or non-CA.

Organizational PKI: The concept of PKI, relying on authorities to certify users, goes much broader than web applications. 1) VPN is a PKI in the sense that the IT department plays the role of CA and users off campus can use their public keys to get authenticated. 2) Credit card organization, like Visa, is a PKI that establishes trust between different banks and their customers. Visa is the CA and banks are the users (so that buyer A in Bank 1 can do business with seller B with account in Bank 2).

There is a dream of universal PKI that a single CA certifies everything for applications across the board. However, in practice, the most successful models are organizational, domain-specific PKIs. Why?

Trust and authorities are essentially context-specific. You trust campus IT to the degree of revealing your SUID. But you don't trust IT when it comes to revealing your credit card number. You trust amazon.com for credit card. It is hard to find a universal CA that everybody can place unlimited trust to.

Names: PKI certifies the binding between a public key and a name. But what about the binding between a name and an actual physical entity (1)? There are other problems regarding names: 2) alias problem that one person has multiple (nick)

names, 3) common names in that different people share the same name. In a small town, problem 1) is solved by everybody knows everybody else. And the name-person binding is solved by sight. Problem 2) and 3) are solved by gossiping. Villagers can propagate names "Big John" and "Little John" to distinguish two Johns. In a large scale setting, we rely on authorities to assign name. Like US government assigns SSN to identify different citizen. Email service provider assigns unique email account for a user. Names and the use of names is context specific (at home he is called John, in workplace, he is called Mr. Smith). This is another reason context-specific PKI is preferred over universal PKI.

PKI and authorization: PKI certifies the binding between names and public keys. ACL or access-control list stores the binding between names and permissions. Indirect authorization couples PKI with ACL to solve the authorization based on public keys. However, the alias problem in names makes it ineffective to join the PKI and ACL; what if the name in PKI is "John" and name used in ACL is "Mr. Smith"?

Direct authorization directly binds public key with permissions. Without name, the alias problem goes away. However, without human-readable names, it makes it difficult to audit.

Credential system is a direct authorization method where the binding between public key and permissions is signed. This sort of certificate makes it easy to transfer permissions. For instance, a file user Alice who has read permission to file X can transfer her "authority" (file read permission) to another user, Bob, by issuing the following certificate: [pk_B is authorized to read file X]; this certificate is signed with Alice's secret key sk_A. The credential system verifies it by 1) verifying the signature against pk_A, 2) checking if user pk_A (namely Alice) has read permission to file X.

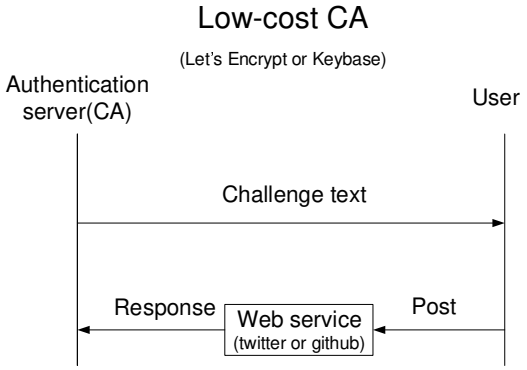

Fig. 10: Low-cost CA

Another example of cryptographically implemented access control is the blockchain. If you think about what happens on a blockchain miner node, it maintains a list of key-value pairs, each with a public key, being the key, and a token/cryptocurrency unit (e.g., Ether) being the value. The binding between the cryptocurrency and public key represents the ownership of cryptocurrency. The key-cryptocurrency pair can be read by anyone (no access control for read) and can be modified only by the secret key associated with the public key (i.e., access controlled for write). The modification here is to remove the current binding and bind the cryptocurrency unit to another public key. In other words, Blockchain's state, ownership of each cryptocurrency unit is a KV pair with public key being the key and cryptocurrency unit being the value. The ownership can also be viewed as an ACL (access control list). The transaction-validator in Blockchain is essentially an access-control engine that enforces the ACL cryptographically; data bound to a public key PK can be updated only when the modify request comes from the secret key SK of PK.

Low-cost CA: Modern CAs like Let's Encrypt and Keybase support low cost. This is done by automation. Figure 10 shows how low-cost CA can piggyback the trust in existing web services (e.g., twitter.com and github.com) to authenticate users.

F. Certificate Transparency (CT)

In a certificate transparency scheme,[3] CA issues certificates and send them to update the log on the log servers. The CT log is an append-only log of web certificates.

A web browser DU verifying a domain N using certificate C audits the CT log to see if C is included in the log. To do so, the log auditor (i.e., web browser) retrieves a proof of inclusion $pf_{\text{include}(C,STH)}$ from the log server. The proof can be used to verify that C is/is not included in the log of STH.

Meanwhile, domain N's owner $DO(N)$ downloads the entire CT log from the log server, say with STH'. She checks 1) if any certificate she requested before is included in the CT log copy under STH', and 2) if any certificate she did not request but still under name N is not there in the log copy under STH'.

To ensure no fork attack, domain owner and auditor need to periodically gossip their STH. Consider domain owner DO has log copy under STH and web browser DU under STH'. DO and DU request a consistency proof from the log server

[3]This subsection is summarized from MIT distributed systems course lecture note: http://nil.lcs.mit.edu/6.824/2020/notes/l-ct.txt

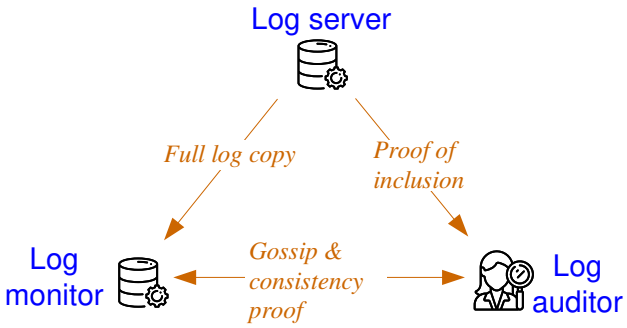

Fig. 11: CT model

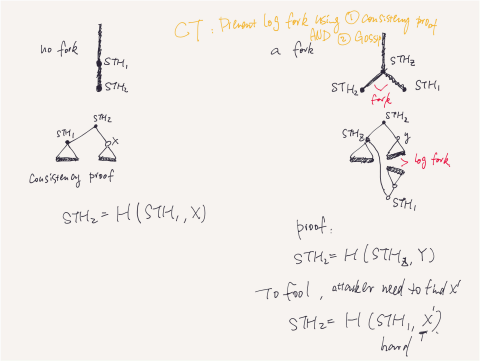

Fig. 12: Forking attacks in CT

to prove that either STH is a prefix of STH' or STH' is a prefix of STH. Only when this consistency proof is verified, there is no fork.

How can CT be attacked: 1. "there is window between when browser uses a cert and monitors check log." 2. "there may be no one monitoring a given name". 3. "lack of gossip in current CT design/implementations". 4. "privacy/tracking: browsers asking for proofs from log server".

XI. Summary

TABLE II: Components in applied cryptography

Scheme/Execution	Private-key enc.	MAC	Public-key enc.	Digital sig.	Protocols
Security definition	EAV,EAV-MUL,KPA,CPA,CCA	ACMA	CPA	ACMA	
Construction	3.17, 3.30	PRF	El Gamal	Schnorr	DHKE
Assumption			DL	DL	DL

REFERENCES

[1] J. Katz and Y. Lindell, Introduction to Modern Cryptography. Chapman and Hall/CRC Press, 2007.

[2] R. B. Lee, Security Basics for Computer Architects, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013. [Online]. Available: https://doi.org/10.2200/S00512ED1V01Y201305CAC025

APPENDIX A
REVIEW: DISCRETE PROBABILITY (§ A)

A. Introduction and Random Variables

Introductory example: Weather forecast: Tomorrow's weather is a random variable that takes different values in its domain (e.g., "rainy", "sunny", "cloudy", etc. This random variable is a distribution today – for each value, there is a probability that the variable takes the value. But on the day after tomorrow, the variable becomes fixed and its value is determined – only one value in the domain is bound with 100% chance.

In abstract, we talk about **(discrete) random variable** R. The random variable R has a domain $\mathcal{R}$ which is a finite set of values r. The variable can take any value in the domain; for each "event" that random variable R takes a value r, denoted as $R \leftarrow r$, it occurs at a certain probability, denoted as $Pr[R \leftarrow r]$. In other words, a random variable is essentially a probability distribution $\mathcal{D}$ defined over the choice of values $r \in \mathbb{R}$. A special case is 0/1-random variable whose finite set is simple $\{0, 1\}$.

Bayes theorem: Bayes theorem describes the probability of an event E_1, based on prior knowledge of conditions E_2 that might be related to the event. That is, $Pr[E_1|E_2] = \frac{Pr[E_2|E_1]}{Pr[E_2]} \cdot Pr[E_1]$. Here $Pr[E_1|E_2]$ is a conditional probability, namely the likelihood of event E_1 occurring given that another event E_2 is true (or occurred). $Pr[E_2|E_1]$ is also a conditional probability, namely the likelihood of event E_2 occurring given that another event E_2 is true (or occurred). $Pr[E_1]$ is the probability of observing event E_1.

B. Statistical Properties

There are several statistical properties describing the distribution of a random variable R. Those include **expectation EXP [R]**, **variance VAR [R]**, and **entropy ENTROPY [R]**. Formally, the expectation of random variable R is $\text{EXP}[R] = \sum_r Pr[R \to r] \cdot r$. When $\text{EXP}[R]$ is known, its bound of the random variable is given by Markov inequality: $Pr[r \leq b] \leq \text{EXP}[R]/b$ for a given bound b. Note that here $Pr[r \leq b]$ is short for $Pr[R \to r \wedge r \leq b]$.

The variance of random variable R is $\text{VAR}[R] = \sum_{r \in \mathbb{R}} (r - \text{EXP}[R])^2$, which can be further derived as $\text{VAR}[R] = \text{VAR}[R] = \text{EXP}[R^2] - \text{EXP}[R]^2$. Consider, for example, a 0/1 random variable R_{01}. We have $\text{VAR}[R_{01}] \leq 1/4$ (proof see book). When $\text{VAR}[R]$ is known, we can use Chebyshev bound to describe its probability bound: $Pr[|R - E[R]| \geq \delta] \leq \frac{\text{VAR}[R]}{\delta^2}$.

The entropy of random variable measures how much uncertain the variable is. Given R, its entropy is $\text{ENTROPY}[R] = \sum_{r \in \mathbb{R}} -Pr[R = r] \cdot \log Pr[R = r]$.

Consider, for example, how to generate a random number? A random number is generated by following a *uniform distribution*. A uniform distribution takes the same probability for each value in its domain, having the expectation being unbiased (being the average) and having a high entropy value. Think about toss a biased coin to generate a random number. The random number is with high entropy (as it is very uncertain which value it will take), but is not uniformly distributed because the coin head is heavier than tail. A solution to this biased tossing is to toss the coin two times, the random variable takes value 0 when the result is head-tail, it takes value 1 when the result is tail-head. Otherwise, it continues.

C. Advanced statistical properties

There are advanced statistical properties, including Chernoff bounds and Birthday problem. A random variable can be best understood in the imaginary experiment of tossing a ball into a set of bins (the balls-in-bins experiment). Those properties are about a random variable taking multiple values or toss the same ball (with same probability) to the same set of bins, *multiple times*. In this setting, Chernoff bounds are about the event that majority of random values is equal to a specific value.

In the multi-toss setting, Birthday problem is about the event that among the multiple values taken, there exist at least two equal values (or collision). Formally, given R,

$$\begin{aligned} \text{COLL}(q, N) &\stackrel{\text{def}}{=} Pr[\text{exist a collision in } q\text{-balls-in-}N\text{-bins}] \\ \text{COLL}(q, N) &\leq \frac{q^2}{2N} \\ &\geq \frac{q^2}{4N} \end{aligned}$$

Consider, for example, the real-world birthday problem that polling about 23 people suffices to find two having the same birthday. That is, when $N = 365, q = \sqrt{365} \approx 23$, $\text{COLL}(q, N) \leq \frac{1}{2}$.

APPENDIX B
BASIC LOGIC OPERATIONS

A. XOR

The truth table of XOR is in Table III.

XOR has a property that $m \oplus m$ can be canceled out in an equation.

TABLE III: XOR truth table

0	$\oplus$	0	0
0	$\oplus$	1	1
1	$\oplus$	0	1
1	$\oplus$	1	0
x	$\oplus$	x	0
0	$\oplus$	x	x