

Lecture 1

21 October 2009

Fall 2009

Scribes: D. Shahaf

1 Introduction

This is a course on cryptography and game theory. We will review basic cryptography (common problems, techniques, and models), but either prior background in the field or independent reading of background materials (such the scribe notes of last year’s “Introduction to Cryptography” course) will be very helpful. No background in game theory is assumed.

Grade composition. If there is a final exam, the grade will be weighted as 30% for the final exam, 30% for the homework assignments, and 40% for the scribe notes. If there is no final exam, the weights will be 45% homework and 55% scribe notes.

Today. We will overview the fields of cryptography and game theory separately (§2.1 and §2.2) and then their combination in (§2.3). In the remaining time we will conduct a quick review of basics of modern cryptography (§3).

2 Overview

2.1 Overview of Cryptography

The philosophy. The goal of cryptography is to protect systems against various attacks, both naturally-occurring and malicious (man-made) ones. A system consists of various *parties*, where a party can be either a computer or a human. A *protocol* (or *scheme*) specifies an algorithm for each party to run, and gives some guarantees to each party that follows its own algorithm — even if the other parties do not fully follow their own instructions. A party is *honest* if it follows the protocol instructions.

Parties who do not follow the protocol instructions may be good-intended but mistaken (as to which protocol to run or software to use), or may be malicious conspirators determined to undermine the honest parties’ plans. Cryptography typically ignores these “fine distinctions” and assumes the worst:

It is typical in cryptography to assume that a single entity, the *adversary*, coordinates all the non-honest parties with a single goal of breaking the protocol’s guarantees to the honest parties. In order to maximize the trust-worthiness of the system, we wish to ensure protection from (resilience to) *arbitrary* attacks, while still maintaining a given functionality to the (honest) users. Towards this end, the adversary is treated as an arbitrary external entity: minimal assumptions are made about it, and the guarantees made should hold in the face of *any* external party meeting these minimal assumptions.

Common minimal assumptions are that the adversary cannot use unlimited computing resources and that the adversary does *not* have full control over the computers used by the honest parties. The adversary is often given complete control over the communications network and is sometimes even allowed to have limited access to the “secret” parts of the system (for example, in encryption scenarios the adversary may be given limited access to the decryption facilities and to the messages being encrypted).

2.1.1 Typical problems

Secure communications. Two parties, A and B , are geographically separated and wish to communicate and maintain secrecy with respect to third parties.

To define this problem, we model all external components as a single adversary and have to define the abilities of this adversary (e.g., how much control/access it has over the communications media) and the goals (guarantees) we wish to achieve (i.e., the concerns we wish to address).

Two commonly-considered types of adversaries in this case are *passive* adversaries — those limited to reading the messages transmitted, but unable to inject or modify messages — and *active* adversaries, which have full control of the channel.

Possible guarantees we might want a secure communications protocol to give are secrecy (the adversary doesn't learn “anything” about the message), authenticity (the receiver of a message can detect when a message was modified by the adversary), and availability (knowing that a sent message will eventually be received).

Note again that the guarantees here are about preserving the interests of the “good guys” — our parties A and B — against *any* third party, regardless of the latter's intent, behavior, or eye color. No guarantees will be made to the third party (cryptography doesn't cater for bad guys), or even to A (resp., B) in the case that B (resp., A) cooperates with the adversary.

Joint computation (the millionaires problem). Two parties, A and B , have inputs x and y respectively. They wish to compute some (agreed upon) function value $f(x, y)$, without revealing any information on their inputs to each other. The canonical use case is when the two parties are millionaires and wish to determine who is the richer of them without leaking any other information about their respective wealths: they would use this primitive with a function $f = f_{\leq}$ that returns a bit indicating which of its two inputs is the larger one.

Unlike in the previous example, now we view the parties not allied in their quest for security (secrecy) against a third party, but rather separated and potentially hostile to each other — wanting to achieve some goal in common, but not trusting each other for anything beyond that. To model this, in multi-party protocols we model some of the players as adversarial. Formally, the external adversary entity is endowed with the ability to *corrupt* some of the parties: upon corruption, the adversary gains complete control of the corruptee and replaces the latter's code with one of his choosing.

The guarantees we would like here are (strong) secrecy — B won't learn anything about x (for example, if $f = f_{\leq}$, B won't learn anything except whether x is larger or smaller than y), and vice versa — and correctness, even against arbitrary (cheating) parties on the other end of the line. We model the situation by letting the adversary control one of the two players, and the protocol's goal is to protect the honest player from badness — in the face of arbitrary behavior by his corrupted (adversarial) opponent.

Upon second thought, however, this formulation of correctness appears to be unenforceable: assuming A is honest, how can he force an arbitrary B to provide the *real* value of y to the join-computation algorithm — rather than some other value y' ? (Put another way, a trivial attack on B 's part is to run the protocol honestly but lie about the status of his bank account.) Indeed, within the scope of this example, there is no such power. Therefore, our security criterion will sidestep this issue: it will not consider a protocol prone to this ‘lying attack’ insecure.

We will say that a joint computation scheme is secure if it emulates an *ideal* process for joint computation: that process consists of the two parties providing their inputs to a common third party — called the *trusted party* — and having the latter compute the parties' outputs

and provide them to the parties. (See [Can06] for more on the “trusted party” style of security definitions.)

The ideal model captures secrecy of the initial inputs, correctness of the computation on the provided inputs, and independence of the provided inputs; therefore, by emulation, a real (concrete) joint computation scheme will have to satisfy the same properties for us to consider it secure.

However, even our ideal joint computation process does not attempt to coerce the parties into providing the “correct” inputs to the computation; the ‘lying attack’ is allowed, and the calling protocol (the one using a joint computation subroutine) will have to implement input validation on its own.

2.1.2 Techniques and modeling

In almost all cases, we have to assume that the adversary is *computationally bounded* in order to have any hope of meeting our security definition. We usually assume that the adversary is limited to probabilistic polynomial-time (PPT) computations: probabilistic, because the adversary should be at least as strong as the honest parties, who *have* to be probabilistic in order to meet our security definitions (even against deterministic adversaries); and polynomial-time, because equating ‘efficient’ and ‘poly-time’ is traditional and convenient. (Other limitations, such as space (storage) complexity, can be considered similarly.)

Likewise, we have to accept a small (“negligible”) probability of error, that corresponds to cases such as the adversary guessing correctly the random coins we used. We will strive only to prove that schemes are secure and correct with extremely high probability over the random coins involved.

Furthermore, proofs of security (even against bounded adversaries) are not absolute, simply because complexity theory hasn’t yet succeeded in showing suitable problems to be not solvable by any efficient (PPT) machine. Therefore, we will take comfort in *conditional* proofs: our security guarantees will be conditional on (done by reduction to) the hardness of computational problems (such as the integer-factoring problem).

Finally, we have to specify the allowed resources (running time) of the adversary and the associated probability of its breaking the system. In order to avoid tedious numerics, we use an asymptotic formulation: we will introduce a *security parameter* (an integer) and investigate the behavior of the system as it tends to infinity. In particular, we will allow all parties to use resources polynomial in the security parameter and require the adversary’s success probability to be *negligible* in the security parameter.

Definition 1.1 *A function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ is negligible if it approaches zero faster than any inverse polynomial; that is, if $f(n) = o(n^{-c})$ for every $c \in \mathbb{N}$.*

2.2 Overview of Game Theory

Game theory is the study of decision making. It is a branch of economics and social science, which is concerned where situations involving multiple ‘players’ (called *agents*) interact with each other and try each to maximize their own utility (profit) through taking actions and decisions. Its goal is to study the connection between situations and their outcomes: what steps and decisions will (should) be taken in a given situation, what other outcomes are possible or desired, how are those related to the ‘rules’ of the interaction, and how to engineer ‘rules’ that lead to given (desired) outcomes.

Like cryptography, the underlying real-life situation in game theory is of many people with conflicting interests attempting to do something in common despite those conflicts.¹ Cryptography only concerns itself with devising protocols to achieve given goals; game theory attempts both to predict the outcomes of given situations (assuming free choice and well-defined goals of the participants) and to design situations that best achieve given goals (see §2.2.3).

2.2.1 Basics of game theory

A basic concept: Games. As the name suggests, a central object in game theory is the *game*. A game consists of a set of participants; each participant is initialized with some information on the other participants, with goals (preferred game outcomes), and with possible actions to take. In general, a game may consist of one move to each player, or of several turns done interactively.

The participants are assumed to be entirely selfish, each acting solely to their own interests; they will not change their behavior to assist or harm another party if they do not gain themselves from doing so.

Naturally, when using games and game theory to study a real-life situation and analyze one's strategies, one ought to carefully validate one's modeling of the situation as a game: the results of the theory will not be valid unless the translation from real-life to a formal 'game' object is faithful.

Another basic concept: Strategy. A *strategy* in a game is the sequence of decisions made by a player. A strategy may be probabilistic (i.e., the player may toss coins to make some of his decisions). It is the game-theoretic equivalent of the notion of 'algorithm' from computer science.

Yet another basic concept: Equilibrium. Say that a player has an incentive to change their play in the game if doing so is expected to increase that player's gain. Now we can define equilibria:

Meta-definition: *An equilibrium in a game is a set of strategies for the game's agents such that no agent (or set of agents) have an incentive to change their strategy, provided that all other agents do not deviate from the equilibrium strategy.*

Example: A simple game This is a *one-shot* game for two parties. (One-shot games are those that where each party has exactly one turn and both parties make their plays simultaneously; then the game ends with a utility (result) for each player. The utility of each player — his gain or loss — depends on the outcome of the game, which depends on the moves of both players.)

In this game, each player has two possible moves: the first player (\mathcal{R}) can play either 'top' (T) or 'bottom' (B), and the second player (\mathcal{C}) can play either 'left' (L) or 'right' (R).

The gains of each player in each game results are given by the following table. For example, in the game variant depicted in Table 1a, if \mathcal{R} plays T and \mathcal{C} plays L, then \mathcal{R} 's gain is '2' and \mathcal{C} 's gain is '0'.

Consider the first game (described in Table 1a). We see that regardless of \mathcal{C} 's play, it is always (strictly) beneficial for \mathcal{R} to play T: for any (fixed) possible play of \mathcal{C} in this game, \mathcal{R} 's gain will be higher if he plays T than if he plays B. Therefore, we predict that \mathcal{R} will play T.

¹Sorry, I didn't mean to sound like a peace-agreement news anchorman.

	L	R
T	(2,0)	(1,2)
B	(0,2)	(0,1)

(a) Original game

	L	R
T	(2,0)	(1,2)
B	(3,2)	(0,1)

(b) Modified game

Table 1: Gains of each player (\mathcal{R} , \mathcal{C}) in each payout of the games in this example

Since \mathcal{C} is “rational”, we predict that she will anticipate \mathcal{R} ’s deductions and his ultimate choice to play T, and will plan her play accordingly. Since \mathcal{C} ’s gain in the payout (T, R) is higher than her gain in the payout (T, L) (the former gain is ‘0’ and the latter gain is ‘2’), we predict that she will play R in order to maximize her gain.

In summary, the theory predicts that the game will play out as ‘(T, R)’. (Note that this is an equilibrium situation: neither player will gain by changing his play, assuming the other player doesn’t change his play.) Under certain assumptions on the players’ reasoning process, this prediction becomes a *recommendation* for each of the players as to which move they should take.

The following definition describes the situation of \mathcal{R} in the last game:

Definition 1.2 (informal) *When a strategy S (for some player P) provides an optimal gain for P regardless of the other players’ strategies, the strategy S is called a dominant strategy.*

Suppose now that we modify the game as shown in Table 1b: the only change is in \mathcal{R} ’s gain for the outcome (B, L). Unlike in the original game, there is no longer a dominant strategy for either player, but the outcome (T, R) is still in an equilibrium. (It is an equilibrium because \mathcal{R} ’s gain if he switches to B, and \mathcal{C} ’s gain if he switches to L, are lower than their respective gains if they remain at their current plays, T and R respectively.)

The equilibrium at (T, R) is not unique, however: in the new game, (B, L) is also an equilibrium. The theory predict that the game will play out as one of the equilibria states. For now we leave open the question of *which* equilibrium will the game play out as.

2.2.2 Two main results of Game Theory

We now state two main existence results (theorems 1.4 and 1.6) and the definitions they require.

Definition 1.3 (informal) *A strategy is mixed if it is probabilistic, i.e., is a distribution over possible plays.*

Theorem 1.4 (Nash, 1950) (informal) *If mixed strategies are allowed, an equilibrium always exists.*

Remark. The theorem’s proof is non-constructive, and no efficient worst-case solution is known for the constructive problem (finding an equilibrium). (The constructive problem is complete for the complexity class PPAD.)

Definition 1.5 (informal) *A correlated equilibrium is an equilibrium in a system where the players have been exposed to a (common) sample from a known distribution. (From a cryptographic standpoint, this is an equilibrium in a system equipped with a common reference string.)*

Theorem 1.6 (Aumann) (informal) [OR94, Prop. 45.3] *Any Nash equilibrium is also a correlated equilibrium, thus correlated equilibria allow for solutions that are at least as good as Nash equilibria. Furthermore, it is possible to compute a correlated equilibrium for any game in polynomial time.*

Remark. There are well-known cryptographic protocols for *joint coin-tossing*: protocols for two parties to execute that allow them to obtain a joint, but unbiased, sample from a given distribution. This suggests a two-party process, whereby first joint coin-tossing is used to manufacture a common reference string, and then the algorithm of Aumann’s theorem is applied to find an equilibrium. Making this idea work is a non-trivial endeavor. We will discuss it in the course.

2.2.3 Mechanism design

So far we presented game theory as a “predictive” discipline, that concentrates on predicting decision making in a given situation. Game theory has also another, more “prescriptive” goal: design games that will lead rational and selfish players into “beneficial interactions”. There can be many different ways to measure the benefits of a payout — e.g., the individual returns, or some more global notions of “social good”. This facet of game theory is called “mechanism design” and the “designed games” are called “mechanisms”.

The quintessential example of a mechanism is an auction: in an auction, we have a seller with some goods, and some buyers who are interested in these goods. An execution of the auction determines how the goods will be distributed and at which price. The goal is to design a game (rules for the auction) that will lead to “good” results. (Here “good” can be “good for the auctioneer”, or “good for the buyers”, or “good for society”.)

2.3 The combination of Cryptography and Game Theory

We see that cryptography and game theory offer two different views on systems where mutually-distrustful individuals interact and attempt to jointly achieve some goals in spite of conflicting interests. The natural question is: How can we combine these two views, and what may we gain by doing so? We offer answers in two directions:

2.3.1 Use Cryptography to advance Game Theoretic goals

We will give some examples of how cryptographic thinking can help the goals of game theory: it can bring the theory’s decisions-making model closer to reality and it help design better mechanisms.

Incorporate computational considerations. As it stands, game theory pays no attention to the computational cost of taking a decision or making a calculation.

For instance, fix a *one-way permutation* f (i.e., a permutation which is provably hard to invert) and consider the game wherein B has two choices: either to participate or not to participate. If B participates, A picks a random x and sends $y := f(x)$ to B , who then has to reply with a value x' . In this game, B ’s utility function gives a small positive gain if he declines to participate, a large positive gain if he participates and sends $x' = x (= f^{-1}(y))$, and a large negative gain if he participates and sends $x' \neq x$.

In traditional game theory, B can always — given y — invert f to obtain x from it, and then submit $x' := x$ as his response and ‘win’ the game; therefore, in the traditional model,

B should always opt to play the game. However, in practice, the “invert f ” step is infeasible to do (in polynomial time), and there is no other way for B to compute the desired answer efficiently (any such way would break the one-wayness of f); therefore, if B opted to play the game, his chances of providing the correct x' would be slim. Hence, in practice, B should prefer the certain “small-but-certain gain” result (of declining to play) over the almost-certain large loss he would suffer if he accepted the game.

More concretely, B 's decision whether or not to accept the challenge of inverting f will depend on the concrete function f , how hard is it to invert for him, and his potential gains from playing the game (with or without inverting f) versus not playing it at all. This way, the “hardness level” of f and the value B assigns to winning the game by inverting f can be used for reasoning about strategies.

We can hope to augment game theory with computational bounds and infeasibility results. Agents will learn to rule out steps which, while information-theoretically are possible, are computationally infeasible.

Enrich equilibria with additional properties. We can characterize equilibria not only by their game-theoretic properties (e.g., the gains of the agents) but also by their cryptographic properties (e.g., how well is the secrecy guaranteed); this way, the agents can take the (quantitative) quality of cryptographic promise into account when making decisions.

Further, having computational bounds at hand will allow us to better analyze the game — in fact, the game itself changes when we introduce the bounds — which may lead, e.g., to better equilibria or better mechanisms (due to the added constraints that can be used in the analysis).

Design better mechanisms. The accumulated cryptographic experience in designing protocols and security criteria could be used to improve the design of games that capture the same situation as the protocol. For example, auctions are treated cryptographically as a commit-then-reveal process (applied to the bids), while game-theoretically each participant tries to balance the cost (of the bid) with the possibility of losing the auction to one of the other bidders.

2.3.2 Use Game Theory to advance Cryptographic protocol design

We saw in the previous paragraphs that sometimes game theory ignores issues of cryptographic hardness. There are also examples of the converse, where cryptography assumes the parties will behave in a non-game-theoretic way.

Example: Secret sharing *Secret sharing* is an n -party task. A message m is picked and n fragments (called *shares*) are derived from it; each share is given to a different party, in a way that enables some sets of parties (but not other sets) to reconstruct the original secret m .

More concretely, a secret sharing scheme labels each set $S \subset [n]$ of parties as either **permitted** or **forbidden**. If a set is **permitted** then its members can reconstruct the secret completely by combining their shares; contrarily, the members of a **forbidden** set should be unable to learn ‘any’ information on the secret, even by pooling all their knowledge.

If the allowed sets are precisely all the sets bigger than some threshold t (i.e., if they are $\{S \subset [n] : |S| \geq t\}$), then a straightforward implementing scheme is known [Sha79]. The case $t = n$ can be solved using a simple system that mirrors a one-time pad (see §3.1).

The protocol to reconstruct the secret is simple: all t parties participating in the reconstruction send each other their shares, and then perform the reconstruction computation locally. However, from a game-theoretic perspective, this protocol calls for irrational behavior: assume that each of the t parties (participating in the reconstruction) wishes to learn the secret himself and to prevent anyone else from learning the secret. Then that party is better off by not sending his own share: by doing so, he will receive the others' $t - 1$ shares and will then be the only one able to reconstruct the secret (since he will be the only one knowing t shares). If all parties follow this logic, the secret will never be reconstructed.

Game theory can be used to avoid designing such protocols, which — as secure as they may be — are irrational and, therefore, wrong at a semantic level (they will never be executed to their end as their author intended them to, assuming the players are rational).

2.3.3 Toward a unified theory

Several issues will arise when we try to combine game-theoretic arguments into cryptographic proofs, or vice versa.

Both cryptography and game theory have their own frameworks and formalisms. To formalize arguments that take advantage of both worlds, it will be desired to have a formalism that can be used to perform both game-theoretic analysis and cryptographic analysis.

For example, cryptography is often interested in security notions that are maintained under *composition*: that is, a protocol that is not only secure when run in isolation, but also remains secure when run in a system with many other (arbitrary) protocols running concurrently to it (by the same or other parties). To combine game-theoretic notions into cryptographic proofs of this kind, we will have to analyze the composability of those notions.

3 Review of Cryptography

We start by reviewing basic concepts in cryptography. Throughout this review we highlight differences between the modeling and formalizations of game theory and cryptography.

3.1 Symmetric encryption

The scenario. The problem of secure (secret) communications dates back at least to the Roman era. We will study several symmetric (private) key encryption schemes and later define security criteria for symmetric encryption.

Classically, we have two (honest) parties, Alice and Bob, and an adversary Eve. The parties' goal is to exchange a message (from Alice to Bob) while maintaining secrecy towards Eve: the latter should not learn anything about the message. (We will formalize the notion of 'anything' later in this section.)

In order for Alice and Bob to have their (necessary) advantage over Eve, we assume they have previously agreed on a key k . (How they agreed is not in scope for the encryption scheme.) Crucially, we assume that Eve has zero a priori knowledge on the key.

Unlike the key, however, the *scheme* (algorithms) that Alice and Bob use are assumed to be public. This standard approach (due to Kerckhoffs) seeks to embed the entire security of the instantiated system in one secret object: the instance's key. Everything else about the system may be publicized.

Let's assume that Eve is passive: she can see everything communicated, but cannot modify or create communications by herself.

Terminology and notation. A *private (symmetric) key encryption scheme* is one where the same *key* is used for both encryption and decryption. The unencrypted message is the *plaintext* and the encryption creates a *ciphertext*, from which decryption with the right key can recover the plaintext again. The *key space* \mathcal{K} is the set of all possible keys, i.e., of all possible instantiations of the scheme. The *plaintext space* and *ciphertext space* are denoted by \mathcal{P} and \mathcal{C} respectively.

We use the abbreviation $[n] = \{1, \dots, n\}$, and we write S_n for the space of permutations on $[n]$. The notation ' $x \stackrel{\$}{\leftarrow} S$ ' means that a random value is chosen (uniformly) from the set S and assigned to the variable x . The operator ' \oplus ' stands for bitwise exclusive or.

When treating encryption schemes formally, we will use the following syntax:

Definition 1.7 (syntax of encryption schemes) An encryption scheme consists of three probabilistic algorithms (Gen, Enc, Dec). (A probabilistic algorithm's input implicitly includes a 'random tape' input that the algorithm may consume).

- The key generation algorithm Gen is a randomized algorithm that returns a uniformly-chosen key $k \in \mathcal{K}$; we write $k \stackrel{\$}{\leftarrow} \text{Gen}()$.
- The encryption algorithm Enc is a randomized algorithm that takes a key $k \in \mathcal{K}$ and a plaintext (aka, message) $m \in \mathcal{P}$ and outputs a ciphertext $c \in \mathcal{C}$; we write $c \stackrel{\$}{\leftarrow} \text{Enc}_k(m)$.
- The decryption algorithm Dec is a randomized algorithm that takes a key $k \in \mathcal{K}$ and a ciphertext (aka, message) $c \in \mathcal{C}$ and outputs a plaintext $m \in \mathcal{P}$; we write $m \stackrel{\$}{\leftarrow} \text{Dec}_k(c)$.
In practice, Dec may be deterministic.

The message space \mathcal{P} is often the set of strings of a given length. (The ciphertext space \mathcal{C} does not have to equal the plaintext space.) We require that $\text{Dec}_k(\text{Enc}_k(m)) = m$ for every k and m as above.

Examples. Below are three classical symmetric-key encryption schemes: Caesar's cipher (shift cipher), substitution cipher, and one-time pad. We assume the plaintext and ciphertext are each ℓ characters long, i.e., $m = m_1 \dots m_\ell$ and $c = c_1 \dots c_\ell$, where each m_i and each c_j is one character from the alphabet. (The plaintext space and ciphertext space may use different alphabets.)

scheme	plaintext space \mathcal{P}	key	$\text{Enc}_k(m)$	$\text{Dec}_k(c)$
Caesar	$\{A, \dots, Z\}^\ell = [26]^\ell$	$k \stackrel{\$}{\leftarrow} [26]^\ell$	$(m_i + k_i) \bmod 26$	$(c_i - k_i) \bmod 26$
substitution	$\{A, \dots, Z\}^\ell = [26]^\ell$	$k \stackrel{\$}{\leftarrow} S_{26}^\ell$	$k(m)$	$k^{-1}(c)$
one-time pad	$\{0, 1\}^\ell$	$k \stackrel{\$}{\leftarrow} \{0, 1\}^\ell$	$m \oplus k$	$c \oplus k$

Table 2: Three classical symmetric encryption schemes

Caesar's cipher was perhaps good for its time, but its main weakness is the small key space — small enough for exhaustive (brute-force) search of the key. Note, however, that this attack requires a non-trivial amount of encrypted material to be available; if, say, only one encrypted letter is available, then this attack fails. (In fact, when used for one letter only, a shift cipher is equivalent to a one-time pad.) Thus, our first lesson is that the key space should be large.

The *substitution cipher* addresses this point by having a key space of size $26! \approx 2^{88}$. Like shift ciphers, substitution ciphers works well for truly short messages; however, if the message is

even slightly long (about a paragraph), then the underlying statistics of the language betray the mapping of the most common letters:

Since the mapping is one-to-one, the most common ciphertext symbol corresponds to the most common plaintext letter, which is likely to be one of the most common letters in English (e.g., **t**, **e**, **s**). Since the statistics of English are well-known, an attacker can easily demap the few commonest and rarest ciphertext symbols, and the way from there to complete decryption (and key recovery) is short. (This technique is known as *frequency analysis*.)

Finally, the *one-time pad* is, in fact, not only immune to both of these attacks, but also to every other attack: it is a perfect cipher. We will formalize this notion below, but intuitively it means that the ciphertext is fully independent of the plaintext (i.e., gives zero information about it) from the attacker's (key-oblivious) point of view.

As appealing as it is, the one-time pad (OTP) falls apart very quickly if misused. Reusing key bits for multiple message bits not only loses the independence guarantee of a perfect cipher, but also makes a successful crack easy. The only known cure is to use a key as long as the message to be transmitted; this, however, is easier said than done, since random bits are not cheap and distributing them secretly is a non-negligible administrative burden.

3.2 Paradigm shift

As we see above, the prevalent design technique in cryptography had been to patch up each attack as it is discovered: substitution ciphers fixed the small-key-space problem of Caesar ciphers; later symmetric ciphers tried to address the frequency-analysis attack by adding more machinery around and instead of the substitution cipher's permutations; and so on. Since the 70's, however, the following paradigm has become standard:

1. *define* the security goals and threats (i.e., the adversary's assumed abilities);
2. *construct* a scheme;
3. *prove* the constructed scheme meets the specified goals.

3.3 Security of encryption

Following this paradigm, we will define security goals for encryption. The first security definition we study was proposed (much ahead of its time) by Claude Shannon in 1948:

3.3.1 Perfect secrecy: The achievable dream

Definition 1.8 (perfect secrecy) *An encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ is perfectly indistinguishable if $\forall m_0, m_1 \in \mathcal{P}$, $\forall c \in \mathcal{C}$, and for $k \leftarrow \text{Gen}()$, the random variables $\text{Enc}_k(m_0)$ and $\text{Enc}_k(m_1)$ are distributed identically; that is:*

$$\text{Prob}[\text{Enc}_k(m_0) = c] = \text{Prob}[\text{Enc}_k(m_1) = c],$$

where the probability is taken over the coins of Gen and Enc .

Remark. This is a strong requirement: it does not force the adversary into one particular route — e.g., discovering the message or the key — but it allows her complete freedom in choosing how to distinguish the two messages; we require nothing beyond the ability to distinguish the encryptions of two adversarially-chosen messages. (Due to the universal quantifier, we can think of the messages as having been chosen by the adversary.)

Remark. The requirement for *equal* distributions is akin to placing no computational bounds on the adversary: there must be no way whatsoever — not even an extremely inefficient way — to distinguish the encryptions of two messages, other than know what key to decrypt the two ciphertexts with. This is analogous to game theory, where computational bounds are ignored and, consequently, any rival entity effectively has unbounded resources (see §2.3.1).

The following famous theorem describes how large a key space is required to achieve perfect secrecy.

Theorem 1.9 (Shannon, 1948) *If an encryption scheme is perfect, then the $|\mathcal{K}| \geq |\mathcal{P}|$.*

Corollary 1.10 *No perfect scheme uses fewer than one key bit per plaintext bit.*

Corollary 1.11 *The one-time pad scheme is optimal: it achieves perfect secrecy with key length equal to the plaintext length.*

3.4 Getting around the information-theoretical limitation

3.4.1 First attempt: Statistical secrecy

Shannon’s theorem implies we have no hope of improving upon the one-time pad while maintaining perfect secrecy. We may try to bypass it by weakening our security definition and replace the perfect secrecy by something that, while theoretically weaker, seems to be as “good enough”.

Definition 1.12 (statistical ϵ -indistinguishability) *Two random variables X, Y with support S are said to be statistically ϵ -indistinguishable if*

$$|\text{Prob}[X \in T] - \text{Prob}[Y \in T]| < \epsilon$$

for every event $T \subset S$.

Definition 1.13 (statistical secrecy) *An encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ is ϵ -statistically-indistinguishable if $\forall m_0, m_1 \in \mathcal{P}, \forall c \in \mathcal{C}$, and for $k \leftarrow \text{Gen}()$, the random variables $\text{Enc}_k(m_0)$ and $\text{Enc}_k(m_1)$ are statistically ϵ -indistinguishable, where the probability is taken over the coins of Gen and Enc .*

Remark. If $\epsilon = 0$ then the definition reduces to the previous definition (perfect secrecy).

Remark. This definition quantifies over all events T , regardless of whether or not they are recognizable by an efficient algorithm. In the next subsection we will study a security definition that addresses the ‘efficiently recognizable’ aspect of the definition.

The intuition for Definition 1.13 is that we allow the adversary to learn *something* from looking at the ciphertexts and testing them for property T ; however, the resulting advantage should be negligible.

Unfortunately, switching to statistical secrecy does not buy us much in terms of key length:

Proposition 1.14 *If an encryption scheme is ϵ -statistically indistinguishable, then $|\mathcal{K}| \geq (1 - \epsilon) |\mathcal{P}|$.*

Since this relaxation is still too strong, we will weaken it further. The next definition will do so by discounting the tests T that are not efficiently computable by the adversary; we will first, however, make a detour to introduce the asymptotic model used by that definition.

3.4.2 Computational secrecy

Asymptotic formalization. Normally we use an asymptotic formulation instead of spelling out the concrete running times and associated success probabilities of the parties and the adversary:

- We pick a security parameter 1^n (expressed in unary for technical reasons). The running time of all involved parties is required to be polynomial in the security parameter.
 - A “feasible” adversary is *any* $\text{poly}(n)$ -time adversary.
 - $(\text{Gen}, \text{Enc}, \text{Dec})$ are probabilistic polynomial time (PPT) Turing machines (for three *fixed* polynomial time bounds).
- \Rightarrow The scheme should be secure against any PPT adversary (even one whose bounding polynomial is larger than those of the scheme’s algorithms). Consequently, as n grows (e.g., as technology improves), the scheme takes much less time to use than to break (because the relative gap increases with n).
- We allow the parties or the adversary to be *non-uniform* — i.e., let them run a different program (code) for each value of the security parameter.
 - The message space in the asymptotic case is allowed to depend on the security parameter, $P = \bigcup_n P_n$, where $P_n = \{0, 1\}^n$ is typical.
 - Finally, we have to specify the allowed success probability of the adversary. We will allow a negligible (see Definition 1.1) success probability: this is a sufficiently rapidly-diminishing amount of success, and it enjoys nice properties (e.g., closure under addition, multiplication, and composition).

Other success probabilities are also conceivable (even constant probability may be relevant in some cases). For example, consider a game-theoretic setting wherein a party faces the choice whether or not to attempt to break a cryptosystem. The party might base its decision on the cost (computing resources, time) of a cracking attempt versus the success probability of the attempt and the marginal gain from a successful attempt. In other words, the party has to balance the cost of attempting to break the cryptosystem with the probability of that attempt failing (in which case the party has to fall back to the ‘honest’ strategy, but still has paid the cost in computing resources).

Definition 1.15 (computational secrecy [GM84]) *An encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ over message space $P = \bigcup_n P_n$ (where all the messages in each P_n have the same length) is (computationally) indistinguishable if \forall (non-uniform) PPT adversary $\mathcal{A} \exists$ negligible function ϵ such that $\forall m_0, m_1 \in \mathcal{P}$,*

$$|\text{Prob}[\mathcal{A}(\text{Enc}_k(m_0)) = 1] - \text{Prob}[\mathcal{A}(\text{Enc}_k(m_1)) = 1]| < \epsilon(n),$$

where the probability is taken over the coins of Gen and Enc , as well as over the coin tosses of \mathcal{A} .

Remark. The adversary \mathcal{A} can be thought of as an *efficient* statistical test conducted on the two ciphertexts. Since we limit \mathcal{A} to probabilistic polynomial-time computations, we (intentionally) ignore any attacks which require super-polynomial resources, since those are, to us, too costly to be of concern for the values of the security parameter used in practice.

This is in sharp contrast to game theory, where we can think of the adversary as having all the information that it could possibly (even very inefficiently) compute from the information it already has.

References

- [Can06] Ran Canetti. Security and composition of cryptographic protocols: A tutorial. Cryptology ePrint Archive, Report 2006/465, 2006. <http://eprint.iacr.org/>.
- [GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.