SE 4472b

Information Security

Week 3

Formal Security Continued

Aleksander Essex

Western Engineering

Fall 2015

Advantage

Recall the goal of the adversary in the EAV game was to win more than 50% of the time. Let's begin by formalizing this idea. Recall B flips a coin and uses its outcome, b, to select message m_b to encrypt. A then makes a guess b', and wins if $b' = b$.

We define A's *advantage* in winning the game as follows:

$$\text{Adv}(A) = \left| Pr(b' = b) - \frac{1}{2} \right|$$

Advantage

Let's examine the three cases emerging from our definition of *advantage*:

- A **wins more than 50% of the time.** Then $\text{Adv}(A) = p - \frac{1}{2}$ for some $p > \frac{1}{2}$. Therefore $\text{Adv}(A) > 0$. We say A has a (non-zero) advantage in winning the game.
- A **wins less than 50% of the time.** Notice the abs bars in the definition. Here we borrow from the theory of computation: if we have an algorithm that we know guesses *wrong* more than 50% of the time, we can output the opposite guess, make us *right* more than 50% of the time. So here being wrong is as helpful as being right, and we can say A has an advantage in winning the game by whatever margin it loses the game.
- A **wins exactly 50% of the time.** $\text{Adv}(A) = 0$. We say A has *no* advantage.

Polynomially-bounded Adversary

So in the single-guess strategy example, given a security parameter k, the adversary has an advantage of $\frac{1}{2^{k-1}}$. It is easy to see that there exists some k' such that for all $k > k'$ we have:

$$\frac{1}{2^{k-1}} < \left|\frac{1}{\text{poly}(k)}\right|$$

That is to say an adversary using the single-guess strategy has *negligible advantage*. But what about some other adversary using some other strategy?

What if we defined an adversary that had a magic box that always gave it the correct guess? Well that doesn't really seem fair. Nor would modeling the adversary as a non-deterministic polynomial-time Turing machine, since then you could solve NP-complete problems in polynomial time.

Polynomially-bounded Adversary

PPT-bounded Adversary. In order to keep things consistent with the kinds of computational resources found in the real-world, we model the adversary as a *probabilistic polynomial-time* Turing machine. Thus we say the adversary is probabilistic polynomially time (PPT) bounded.

Modeling real-world attackers. This is a pretty reasonable model: if you double the key space of your encryption scheme, it may be possible for the adversary to double its computing power to keep up. But if you double it again, and then double it again... well eventually a real-world adversary is going to run out of money/space/electricity to keep up with you.

We can define *negligibility*, therefore, in terms of how the keyspace grows relative to the adversary's advantage.

Negligible Advantage

Advantage of a ``"single-guess" strategy. Let's suppose A has an advantage winning a game, but it's small. What if it's really really small? Like, so small we can effectively ignore it?

For example, suppose B was using Enigma for encryption, and brute forcing took approximately 2^{76} operations. Suppose A's strategy in the EAV game was to make a single guess about which key B used to encrypt m_b. If the key was correct, A would recover b, otherwise A would make a random case.

A would guess correctly with probability

$$Pr(b' = b) = \frac{1}{2^{76}} \cdot 1 + (1 - \frac{1}{2^{76}}) \cdot \frac{1}{2}$$

and thus

$$\text{Adv}(A) = \frac{1}{2^{77}}$$

Negligible Advantage

In the previous slide we saw that A's single-guess strategy had a small advantage; $\frac{1}{2^{77}}$. But that's a pretty small number.

An inconsequential advantage. Technically A's advantage is non-zero, so you *could* say A has an advantage. But even though this advantage is non-zero, we can characterize it as *negligible*, since it's no better than brute force (which, recall, we said was our worst-case scenario).

Furthermore, A's advantage under this strategy is related to the size of the key space. So if A used the same strategy on 128-bit AES instead of Enigma, A's advantage would be $\frac{1}{2^{129}}$.

So what we need is a definition of ``*negligible advantage*'' that takes the keyspace in to account.

Negligible Advantage

Let security parameter k characterize the size of the keyspace of an encryption function (e.g., k is the key size in bits).

A negligible function. We define $\varepsilon(k)$ to be a *negligible function* in security parameter k if, for any polynomial function poly() there exists some k' such that for all $k > k'$ we have:

$$\varepsilon(k) < \left|\frac{1}{\text{poly}(k)}\right|$$

In other words, $\varepsilon()$ is a negligible function if it grows more slowly than the inverse of any polynomial function. Finally, we say an adversary A has a *negligible advantage* if

$$\text{Adv}(A) = \varepsilon(k)$$

IND-EAV Security

Now we're ready to formally define our basic security notion.

Definition 1 (IND-EAV Secure)**.**

We say a cryptosystem is *indistinguishable under eavesdropping* (*i.e.*, IND-EAV secure) if there exists *no* PPT-bounded adversary with a non-negligible advantage of winning the EAV game.

Notice this security definition captures *all* PPT-bounded adversaries, and thus all PPT-bounded strategies/algorithms you might try. The intuition is that if no such strategy exists, then the encryption system does not leak any practically usable information.

Chosen Plaintext Attack (CPA) Game

IND-EAV is a good starting point as a security notion, but it's not really widely used. That's because, in essence, it's too strong of an assumption to believe *eavesdropping* is the best an attacker could do in practice.

Attacker can encrypt. Let's define a new game that is a little more realistic in terms of what an attacker could do. This new game is almost identical to the EAV game, except we will give the adversary a new *power*: the ability to encrypt plaintexts.

A reasonable assumption. But wait a second, is this *really* a realistic assumption? Well, if we're talking about public-key cryptography, absolutely. By definition *anyone* can encrypt, though only the key holder can decrypt. We'll get to this later in the course.

Chosen Plaintext Attack (CPA) Game

In terms of ``regular'' (*i.e.*, symmetric-key) encryption, it's also reasonable to model the adversary as having the power to encrypt. At very least, we can assume they know something about the structure of a message.

Chosen-plaintexts in history. For example during WWII the Allies would feed the Germans disinformation about military plans. Soldiers would report the fake plans back to headquarters using Enigma, providing the Allies with a chosen-plaintext pair (*i.e.*, a chosen plaintext and its corresponding ciphertext).

Chosen-plaintexts today. In today's Web, there are many opportunities to inject chosen-plaintexts in symmetric-key ciphers. We'll look at this later in the context of padding-oracle attacks.

Chosen Plaintext Attack (CPA) Game

The chosen-plaintext attack (CPA) game proceeds the same as the EAV game, except there is a *query* stage in which A can send messages to B, and B will respond with their respective encryptions.

PPT bounded queries. Once again, modeling A as a PPT-bounded adversary means that we limit the number of queries A can make to being polynomial in the security parameter. So if the key space is 2^k, we can't get away with saying ``suppose the adversary makes 2^k encryption queries,'' since that's basically assuming unnatural ability to scale resources (*i.e.*, it's cheating).

The CPA Game

Public inputs: Encryption function $E()$ and key length s.

A B

The CPA Game

Public inputs: Encryption function $E()$ and key length s.

A	B
	Generate random key $k \in_r \{0,1\}^s$

The CPA Game

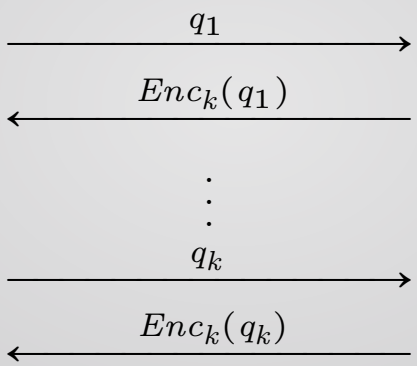
Public inputs: Encryption function $E()$ and key length s.

A

Make polynomially many queries $q_1 \ldots q_k$

B

Generate random key $k \in_r \{0,1\}^s$



The CPA Game

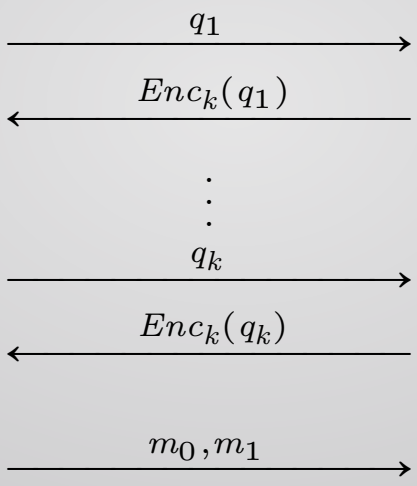
Public inputs: Encryption function $E()$ and key length s.

A

B

Generate random key $k \in_r \{0,1\}^s$

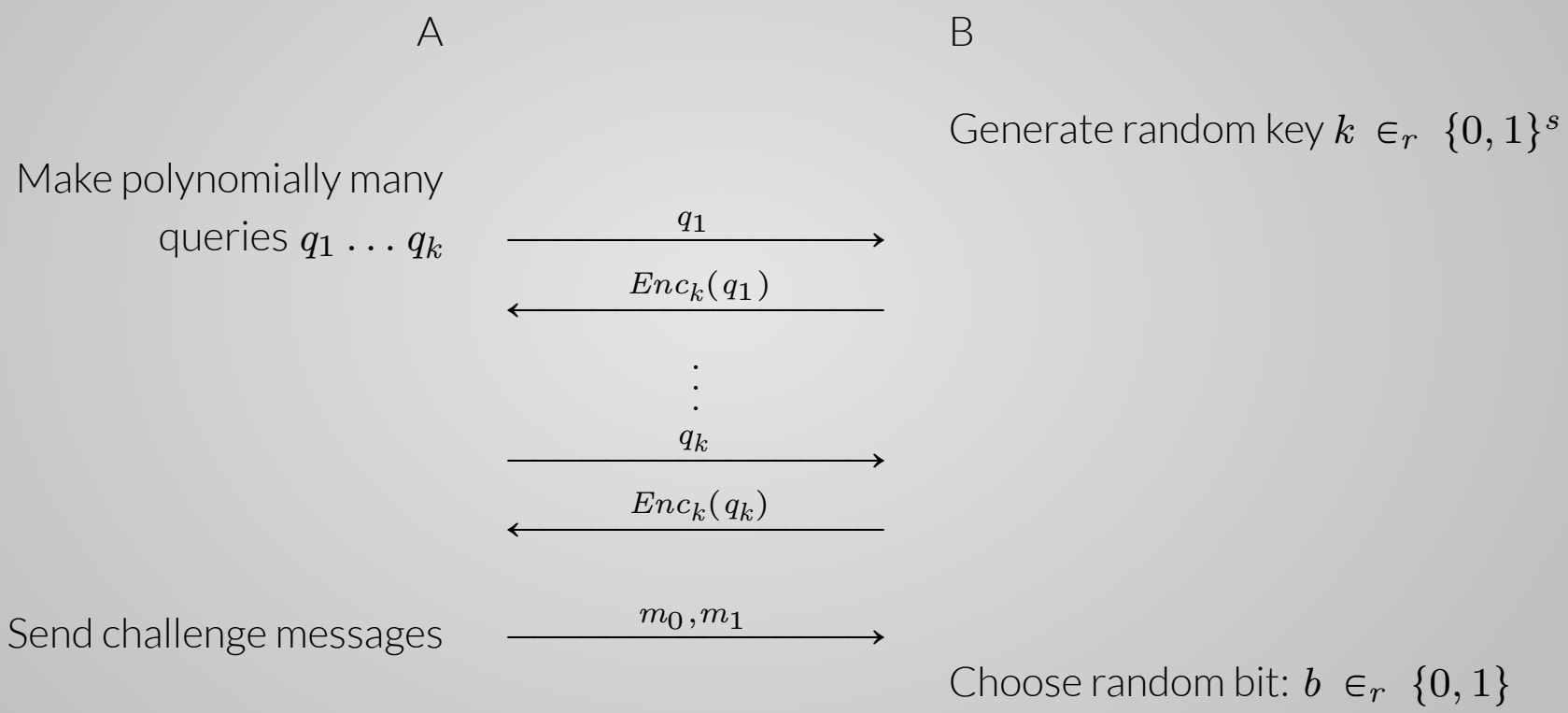
Make polynomially many queries $q_1 \ldots q_k$



Send challenge messages m_0, m_1

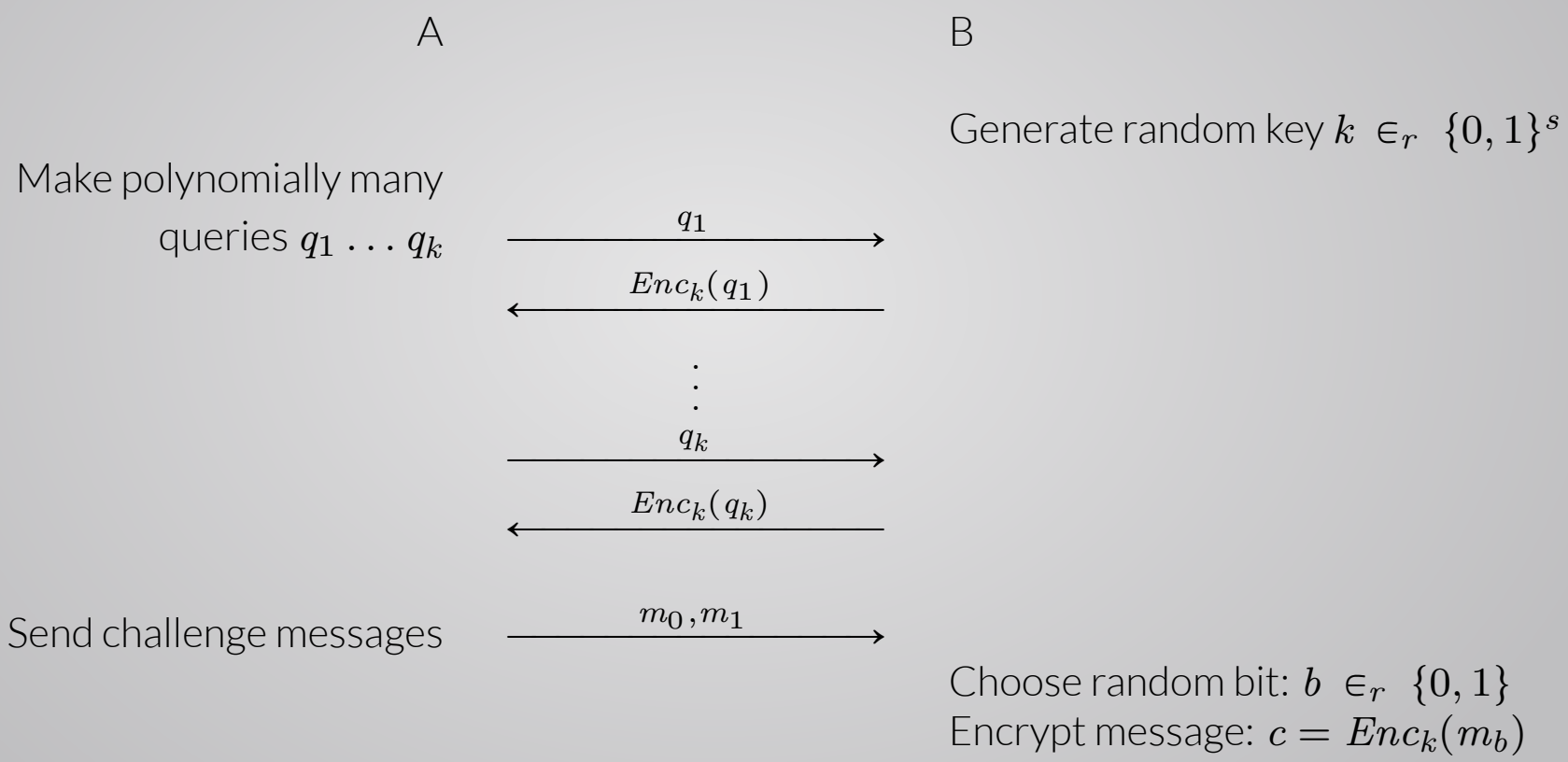
The CPA Game

Public inputs: Encryption function $E()$ and key length s.



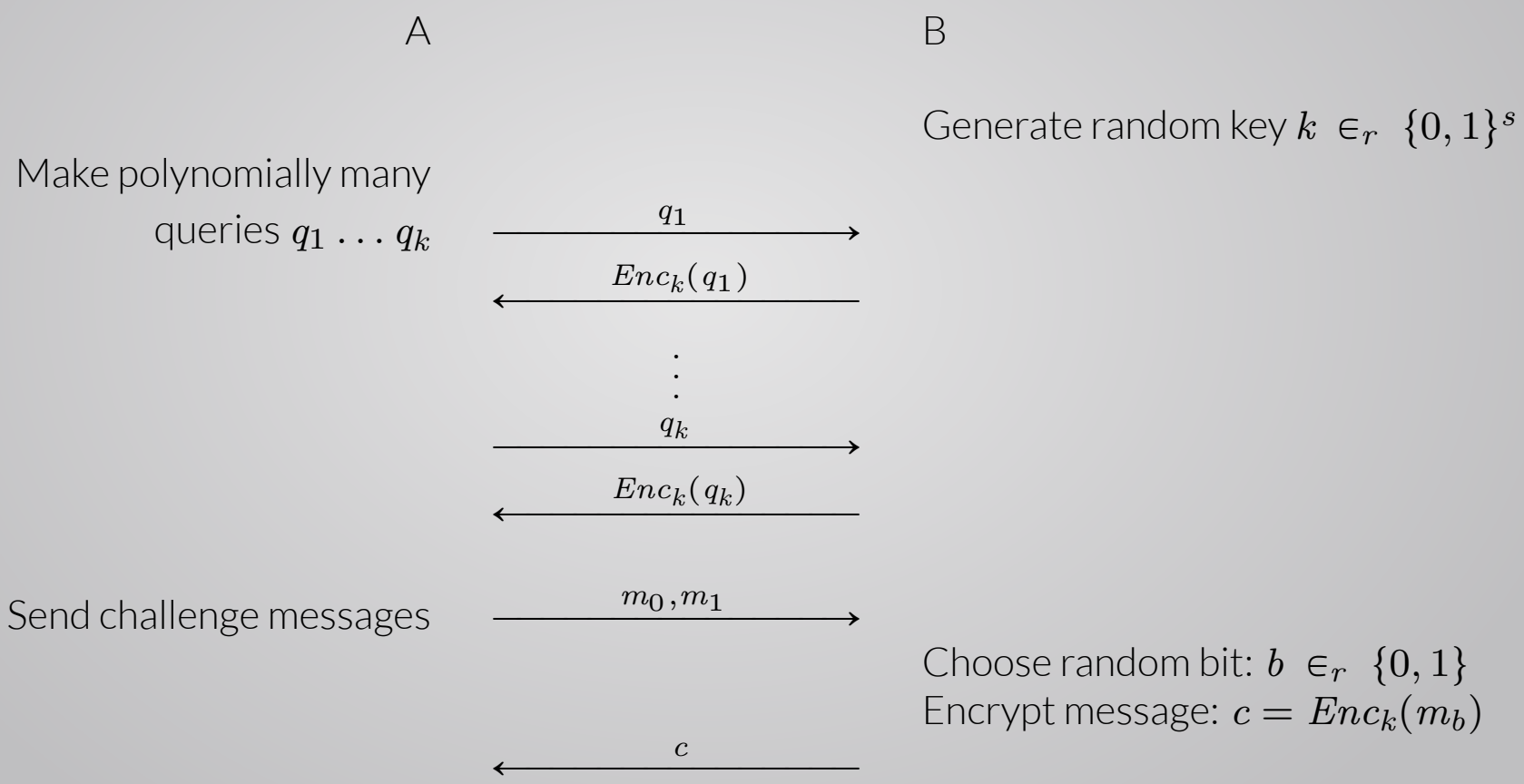
The CPA Game

Public inputs: Encryption function $E()$ and key length s.



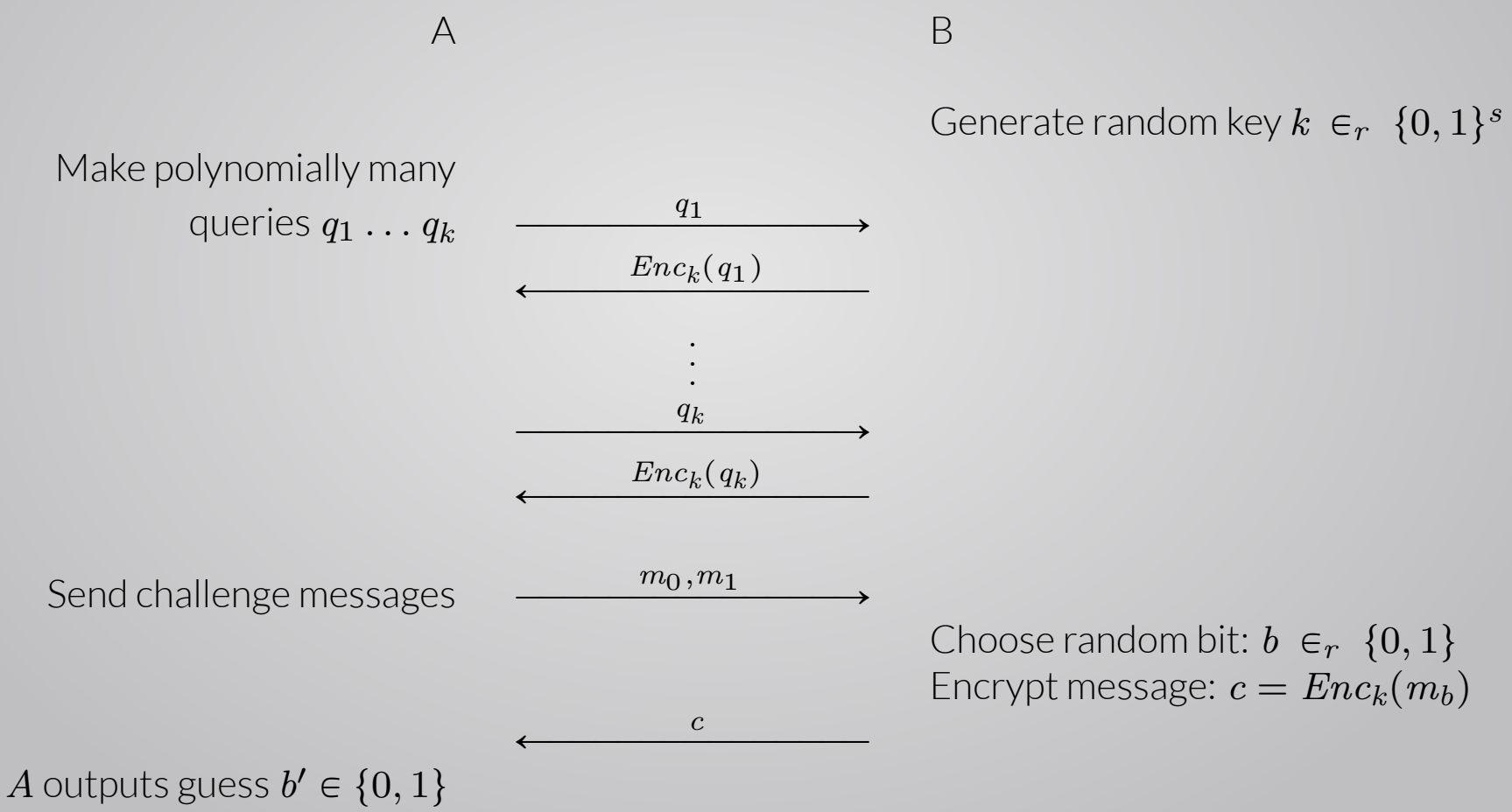
The CPA Game

Public inputs: Encryption function $E()$ and key length s.



The CPA Game

Public inputs: Encryption function $E()$ and key length s.



IND-CPA Security

Definition 2 (IND-EAV Secure)**.**

We say a cryptosystem is *indistinguishable under chosen plaintext attack* (*i.e.*, IND-CPA secure) if there exists *no* PPT-bounded adversary with a non-negligible advantage of winning the CPA game.

Non-deterministic Encryption

Deterministic encryption. What happens if you encrypt the same message twice using the same key? Well for example, if you're using the Caesar cipher, you get the *same* ciphertext both times. We call this kind of encryption *deterministic*, *i.e.*, the key and the message determine the ciphertext.

Winning the CPA game. Suppose $E()$ is a deterministic encryption function used in a CPA game. Suppose A sends B message m_0, during the query phase. A would get back $c_0 = Enc_k(m_0)$. Now if A sends m_1, A would get back $c_1 = Enc_k(m_1)$. Now suppose A sends m_0, m_1 during the challenge phase. B picks one message at random, m_b, and sends back $c_b = Enc_k(m_b)$. Since E is deterministic, all A has to do is check if $c_b = c_0$ or $c_b = c_1$ and output b' accordingly. In this case, A would always win. Deterministic ciphers, therefore, *cannot* be IND-CPA secure.

Non-deterministic Encryption

Non-deterministic encryption. What happens if you encrypt the same message twice using the same key? Well if you're using *non-deterministic encryption* then you essentially get a *different* ciphertext each time.
How does that work? We won't get in to that too deeply at this point. But the basic idea is that a non-deterministic encryption function takes *three* things: a key, a plaintext, and a random value. The ciphertext is a function of all three things, but the decryption function is structured such that the random value disappears during decryption (and the plaintext can be recovered).

We'll see some examples of this later, but the first IND-CPA secure cryptosystem was due to Goldwasser and Micali.

Chosen Ciphertext Attack

In the CPA game we let the adversary make encryption queries *i.e.*, the adversary can *choose* plaintexts and have them encrypted.

Decryption queries. Let's take this idea to its natural conclusion and examine what happens if we also let the adversary *choose* ciphertexts and have them decrypted.

Naturally we call this a *chosen ciphertext attack*. The chosen ciphertext attack (CCA) game is played exactly as the CPA game, but the attacker can also make decryption queries.

Real-world motivation. The idea here is to imagine a scenario where an attacker is able to mount an attack where some *some* messages get decrypted. The question is: what does this reveal (if anything) about other the content of other, future ciphertexts?

Ciphertext Validity

Took keep things interesting (*i.e.*, non-trivial) we will place a couple of conditions on the adversary's ability to make decryption queries:

- **No cheating!** B will refuse to decrypt the challenge ciphertext, *i.e.*, $c_b = Enc(m_b)$. Obviously if we didn't explicitly forbid this, the adversary would always win! Importantly, however, A is allowed to generate new ciphertexts c' that are based on c_b. So, for example, you could do something like set c' to c_b concatenated with 0. B will decrypt c' so long as it's not c_b, and it's a valid ciphertext...
- **Only valid ciphertexts allowed.** The idea here is *some* strings can be decrypted, but potentially there exist some strings that can't (*i.e.*, there is no associated plaintext). Thus some strings constitute valid ciphertexts, and some aren't. We enforce that B will only decrypt *valid* ciphertexts.

Adaptive Chosen Ciphertext Attacks

When we give the adversary the ability to make decryption queries, it comes in two varieties:

1. **Non-adaptive chosen ciphertext attack (CCA1)**. The adversary can make decryption queries *until* the challenge ciphertext is issued. B will decrypt any valid ciphertexts. An encryption scheme is **IND-CCA1** secure if no adversary has an advantage winning the CCA1 game.
2. **Adaptive chosen ciphertext attack (CCA2)**. The adversary can continue to make decryption queries *after* the challenge. B will decrypt any valid ciphertexts, but not the challenge itself. An encryption scheme is **IND-CCA2** secure if no adversary has an advantage winning the CCA2 game.

Adaptive Chosen Ciphertext Attacks

1. **IND-CCA1.** The CCA1 game has been referred to as a *lunchtime attack*. Imagine an attacker makes queries at your (desktop) computer while you are away for lunch. The IND-CCA1 security level is interesting to cryptographers for studying what can be learned about the decryption key given from chosen ciphertexts, but is often not of practical importance.
2. **IND-CCA2.** The CCA2 game is the highest security level we will consider, as it is the one that best protects against the range of real-world attacks on the internet.

Comparison of Security Levels

Summary of the adversary's capabilities at various security levels:

Security level	**Permitted Queries**	
	Pre-challenge	Post-challenge
IND-CCA2 *Adaptive chosen-ciphertext*	Encryption *and* Decryption	Encryption *and* Decryption
IND-CCA1 *Chosen-ciphertext*	Encryption *and* Decryption	Encryption only
IND-CPA *Chosen-plaintext*	Encryption only	Encryption only
IND-EAV *Eavesdropping*	n/a	n/a

Notice each security level inherits the adversarial capabilities of the level below, and thus the security level below. An IND-CCA2 secure scheme, therefore, is also IND-CCA1 and IND-CPA and IND-EAV secure.

Example: Prove Enigma is not IND-CCA2

An example: Prove Enigma is not secure against adaptive chosen-ciphertext attack, *i.e.*, not IND-CCA2 secure:

- **Method 1: Using chosen plaintexts.** You could start by proving Enigma wasn't IND-CPA secure. A chooses two messages m_0, m_1, makes encryption queries for them both, gets their respective ciphertexs c_0, c_1. Then A submits m_0, m_1 and gets challenge ciphertext c_b. A checks if $c_b = c_0$ or $c_b = c_1$. So it's not IND-CPA secure because A always wins, and since the CPA game is a subset of the CCA2 game, its not IND-CCA2 secure.

Example: Prove Enigma is not IND-CCA2

- **Method 2: Using chosen ciphertexts**. Suppose A sends m_0, m_1 and gets back challenge c_b. A can't ask for c_b to be decrypted (that's cheating!) but *can* submit a related ciphertext to be decrypted (as long as its valid).

  Suppose $m_0 = AAA$ and $m_1 = BBB$ and $c_b = Enc(BBB) = FKX$. A could just add another letter to the end of c_b and submit $c'_b = FKXX$ to be decrypted. It's not cheating since $c'_b \neq c_b$, and it's a valid ciphertext (in the case of Enigma any string of letters is).

  Finally the decryption of c'_b would help us deduce the decryption of the challenge c_b. In this case the decryption of c'_b would simply be m_b plus some extra letter e.g., $BBBM$. Once again A always wins.