Radbound University

Recap of Introduction to Cryptography

Applied Cryptography – Spring 2024

Bart Mennink

January 29, 2024

Institute for Computing and Information Sciences
Radboud University

Outline

Course Organization

Security (NWI-IPC021)	$\longrightarrow$	Intro to Cryptography (NWI-IBC023)	$\longrightarrow$	**Applied Cryptography (NWI-IMC061)**	$\longrightarrow$	Cryptology (NWI-IMC063)

Security (NWI-IPC021)	$\longrightarrow$	Intro to Cryptography (NWI-IBC023)	$\longrightarrow$	**Applied Cryptography (NWI-IMC061)**	$\longrightarrow$	Cryptology (NWI-IMC063)

Goal of the Course

- Learn what cryptography is used in applied settings
  - What is used in the real world
  - What is standardized
  - What will (?) be used in the future
- Prepare you for cryptographic aspects you might see later in your career

Feedback Welcome!

- This is the third time the course Applied Cryptography is taught
- We carefully discussed the topics of Applied Cryptography
  - among ourselves
  - with lecturers of earlier courses

Feedback Welcome!

- This is the third time the course Applied Cryptography is taught
- We carefully discussed the topics of Applied Cryptography
  - among ourselves
  - with lecturers of earlier courses

- This course is aimed to complement earlier courses, with minimal overlap
- However, there have been slight mutations in the content of the earlier courses

Feedback Welcome!

- This is the third time the course Applied Cryptography is taught
- We carefully discussed the topics of Applied Cryptography
  - among ourselves
  - with lecturers of earlier courses

- This course is aimed to complement earlier courses, with minimal overlap
- However, there have been slight mutations in the content of the earlier courses
- This means that a minimal overlap with earlier courses is unavoidable

Feedback Welcome!

- This is the third time the course Applied Cryptography is taught
- We carefully discussed the topics of Applied Cryptography
  - among ourselves
  - with lecturers of earlier courses

- This course is aimed to complement earlier courses, with minimal overlap
- However, there have been slight mutations in the content of the earlier courses
- This means that a minimal overlap with earlier courses is unavoidable

- If you have feedback on the course, please contact the lecturers!

Reflection On Last Year

- Course reasonably well-graded
- Some start-up problems identified by students and lecturers

Reflection On Last Year

- Course reasonably well-graded
- Some start-up problems identified by students and lecturers

- Lectures:
  - Further refinement with "Introduction to Cryptography" and "Cryptology"
  - More explanation on how cryptographic functions are used in practice
  - Further overall improvement of applications

Reflection On Last Year

- Course reasonably well-graded
- Some start-up problems identified by students and lecturers

- Lectures:
  - Further refinement with "Introduction to Cryptography" and "Cryptology"
  - More explanation on how cryptographic functions are used in practice
  - Further overall improvement of applications
- Tutorials/Assignments:
  - Make the assignments clearer
  - Less work-intensive assignments

Who?

Lecturers

- Bart Mennink, M1 3.05, `b.mennink@cs.ru.nl`
- Simona Samardjiska, M1 03.18, `simonas@cs.ru.nl`

Assignment Coordinators

- Mario Marhuenda Beltrán, M1 03.17, `mario.marhuendabeltran@ru.nl`

Tutorial Assistant

- Maximilian Pohl, `maximilian.pohl@ru.nl`

Lectures

- Weekly: Mon 13.30–15.15 in HG00.514
  - 5 lectures on symmetric cryptography (Bart Mennink)
  - 5 lectures on public-key/post-quantum cryptography (Simona Samardjiska)
  - 2–2.5 lectures on selected topics (guest lectures)
  - 0.5–1 back-up/Q&A
- Exception: lecture **upcoming** Wed 10.30–12.15 instead of next week Monday

Lectures

- Weekly: Mon 13.30–15.15 in HG00.514
  - 5 lectures on symmetric cryptography (Bart Mennink)
  - 5 lectures on public-key/post-quantum cryptography (Simona Samardjiska)
  - 2–2.5 lectures on selected topics (guest lectures)
  - 0.5–1 back-up/Q&A
- Exception: lecture **upcoming** Wed 10.30–12.15 instead of next week Monday
- Presence not compulsory...
  - ...but if you are going to come, actually be here!
  - Laptops shut, phones away

Lectures

- Weekly: Mon 13.30–15.15 in HG00.514
  - 5 lectures on symmetric cryptography (Bart Mennink)
  - 5 lectures on public-key/post-quantum cryptography (Simona Samardjiska)
  - 2–2.5 lectures on selected topics (guest lectures)
  - 0.5–1 back-up/Q&A
- Exception: lecture **upcoming** Wed 10.30–12.15 instead of next week Monday
- Presence not compulsory...
  - ...but if you are going to come, actually be here!
  - Laptops shut, phones away
- Course material:
  - These slides
  - Lecture recordings

Lectures

- Weekly: Mon 13.30–15.15 in HG00.514
  - 5 lectures on symmetric cryptography (Bart Mennink)
  - 5 lectures on public-key/post-quantum cryptography (Simona Samardjiska)
  - 2–2.5 lectures on selected topics (guest lectures)
  - 0.5–1 back-up/Q&A
- Exception: lecture **upcoming** Wed 10.30–12.15 instead of next week Monday
- Presence not compulsory...
  - ...but if you are going to come, actually be here!
  - Laptops shut, phones away
- Course material:
  - These slides
  - Lecture recordings
- Background material:
  - Lecture notes "Introduction to Cryptography"

Tutorials/Assignments

- Weekly: Wed 10.30–12.15 in E1.17/EOSN01.560
  - 3 assignments on symmetric cryptography (after lectures 2, 3, 5)
  - 3 assignments on pk/pq cryptography (after lectures 7, 8, 11)
  - 1 assignment on selected topics (after lecture 12)

Tutorials/Assignments

- Weekly: Wed 10.30–12.15 in E1.17/EOSN01.560
  - 3 assignments on symmetric cryptography (after lectures 2, 3, 5)
  - 3 assignments on pk/pq cryptography (after lectures 7, 8, 11)
  - 1 assignment on selected topics (after lecture 12)
- Schedule:
  - New assignments on the web by **Monday evening**
  - Two tutorials for asking questions
  - Hand-in: **Sunday after second tutorial, before 23.59** via Brightspace
    - In LaTeX, as single pdf
    - Hint: you are allowed to hand in earlier!
  - General rule: too late means score 0, no exceptions

Tutorials/Assignments

- Weekly: Wed 10.30–12.15 in E1.17/EOSN01.560
  - 3 assignments on symmetric cryptography (after lectures 2, 3, 5)
  - 3 assignments on pk/pq cryptography (after lectures 7, 8, 11)
  - 1 assignment on selected topics (after lecture 12)
- Schedule:
  - New assignments on the web by **Monday evening**
  - Two tutorials for asking questions
  - Hand-in: **Sunday after second tutorial, before 23.59** via Brightspace
    - In LaTeX, as single pdf
    - Hint: you are allowed to hand in earlier!
  - General rule: too late means score 0, no exceptions
- Assignment gives up to 1 point (out of 10) bonus on exam
- Assignments can be handed in in pairs (strongly encouraged)

Organization

Assessment

- Final mark is computed from:
  - Average of markings of assignments: A
  - Open-book on-campus exam: E
  - Final mark: $F = E + \frac{A}{10}$
- To pass: $E \geq 5$ and $F \geq 6$

Further Information

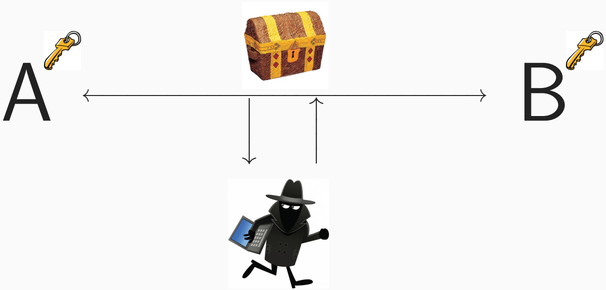
- All information on the course appears on Brightspace
- Read the course manual!

Keyed Symmetric Cryptography

General Setting

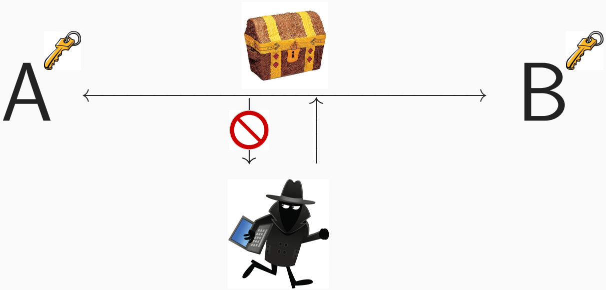
- Two parties, Alice and Bob, communicate over a public channel
  - They have agreed on a joint key and use it to transmit data

General Setting



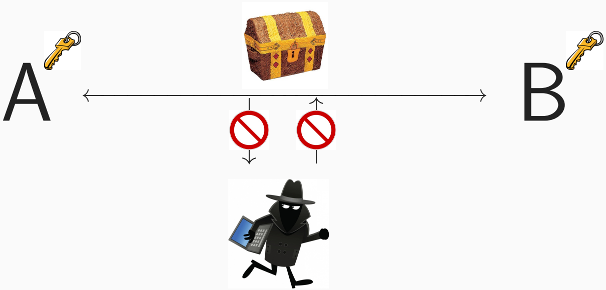
- Two parties, Alice and Bob, communicate over a public channel
  - They have agreed on a joint key  and use it to transmit data
- A malicious party, Eve, may try to exploit/disturb/... the communication
- In symmetric cryptography, we are concerned with two main security properties:

General Setting



- Two parties, Alice and Bob, communicate over a public channel
  - They have agreed on a joint key and use it to transmit data
- A malicious party, Eve, may try to exploit/disturb/... the communication
- In symmetric cryptography, we are concerned with two main security properties:
  - **Confidentiality (or data privacy):** Eve cannot learn anything about data

General Setting



- Two parties, Alice and Bob, communicate over a public channel
  - They have agreed on a joint key  and use it to transmit data
- A malicious party, Eve, may try to exploit/disturb/... the communication
- In symmetric cryptography, we are concerned with two main security properties:
  - **Confidentiality (or data privacy):** Eve cannot learn anything about data
  - **Authenticity:** Eve cannot manipulate the data

Core Functionalities

Encryption

- Uses key to transform data into ciphertext
- Only with the key, one can retrieve data back

Message authentication

- Uses key to complement data with a tag
- Only with the key, the tag can be verified

Authenticated encryption

- Combines encryption and authentication
- Uses key to transform data into ciphertext and tag
- Only with the key, the tag can be verified and data retrieved

Core Functionalities

Encryption

- Uses key to transform data into ciphertext
- Only with the key, one can retrieve data back

Message authentication

- Uses key to complement data with a tag
- Only with the key, the tag can be verified

Authenticated encryption

- Combines encryption and authentication
- Uses key to transform data into ciphertext and tag
- Only with the key, the tag can be verified and data retrieved

These (together with **hashing**) are the core functionalities in symmetric cryptography!

Core Functionalities

- Symmetric stands for:
  - same key for encryption and decryption
  - same key for MAC generation and verification
  - same key for authenticated encryption and verified decryption
  - (cryptographic hashing is an odd one out)

Core Functionalities

- Symmetric stands for:
  - same key for encryption and decryption
  - same key for MAC generation and verification
  - same key for authenticated encryption and verified decryption
  - (cryptographic hashing is an odd one out)

- Throughout, I will assume Alice and Bob managed to share a secret key in such a way that no outsider knows this key

Core Functionalities

- Symmetric stands for:
  - same key for encryption and decryption
  - same key for MAC generation and verification
  - same key for authenticated encryption and verified decryption
  - (cryptographic hashing is an odd one out)

- Throughout, I will assume Alice and Bob managed to share a secret key in such a way that no outsider knows this key
  - This is a problem on its own!

Security Strength s

- Nothing is unbreakable!

Security Strength s

- Nothing is unbreakable!
- Strength of a cryptographic construction is typically measured in bits

Security Strength s

- Nothing is unbreakable!
- Strength of a cryptographic construction is typically measured in bits
- E.g., s bits of security means:
  - there are no successful attacks in less than 2^s operations

Security Strength s

- Nothing is unbreakable!
- Strength of a cryptographic construction is typically measured in bits
- E.g., s bits of security means:
  - there are no successful attacks in less than 2^s operations
  - the success probability of one attempt is at most $\mathbf{Pr}\,(\text{success}) \leq 1/2^s$

Security Strength s

- Nothing is unbreakable!
- Strength of a cryptographic construction is typically measured in bits
- E.g., s bits of security means:
  - there are no successful attacks in less than 2^s operations
  - the success probability of one attempt is at most $\mathbf{Pr}(\text{success}) \leq 1/2^s$
  - generalization: the success probability of an attack with 2^a operations is at most $\mathbf{Pr}(\text{success}) \leq 2^a/2^s$

Security Strength s

- Nothing is unbreakable!
- Strength of a cryptographic construction is typically measured in bits
- E.g., s bits of security means:
  - there are no successful attacks in less than 2^s operations
  - the success probability of one attempt is at most $\mathbf{Pr}(\text{success}) \leq 1/2^s$
  - generalization: the success probability of an attack with 2^a operations is at most $\mathbf{Pr}(\text{success}) \leq 2^a/2^s$
- Refinements often in:
  - data complexity: amount of observed data (limited by use case)
  - computation complexity: amount of computation (limited by budget)

Security is mainly determined by three factors:

1

Length of the key

What Determines Security?

Security is mainly determined by three factors:

1

Length of the key

2

$$K \longrightarrow \boxed{\text{primitive}} \longrightarrow Z$$

Security of the function
that handles the key
(relative to the amount of uses)

Security is mainly determined by three factors:

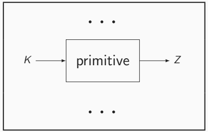
1

Length of the key

2

Security of the function
that handles the key
(relative to the amount of uses)

3



Method in which that
function is employed

How Are Symmetric Cryptographic Schemes Built?

- Building blocks: primitives
  - Determines security factors 1 and 2
  - These are often fixed size functionalities

How Are Symmetric Cryptographic Schemes Built?

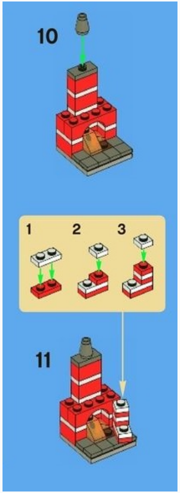
- Building blocks: primitives
  - Determines security factors 1 and 2
  - These are often fixed size functionalities
- Constructions or modes of use employ primitives to build a cryptographic scheme
  - Determines security factor 3
  - Often, these should process variable-length data
  - Constructions not always trivial

How Are Symmetric Cryptographic Schemes Built?

- Building blocks: primitives
  - Determines security factors 1 and 2
  - These are often fixed size functionalities
- Constructions or modes of use employ primitives to build a cryptographic scheme
  - Determines security factor 3
  - Often, these should process variable-length data
  - Constructions not always trivial
- Distinction is a bit fuzzy:
  - Cryptographic schemes themselves are often employed in cryptographic protocols
  - Constructions from one primitive may be primitives for another construction

How Are Symmetric Cryptographic Schemes Built?

- Building blocks: primitives
  - Determines security factors 1 and 2
  - These are often fixed size functionalities
- Constructions or modes of use employ primitives to build a cryptographic scheme
  - Determines security factor 3
  - Often, these should process variable-length data
  - Constructions not always trivial
- Distinction is a bit fuzzy:
  - Cryptographic schemes themselves are often employed in cryptographic protocols
  - Constructions from one primitive may be primitives for another construction



- Symmetric cryptographic schemes on number-theoretical problems exist, but are hardly ever practical

Provable Security

- Symmetric cryptographic schemes on number-theoretical problems exist, but are hardly ever practical
- Symmetric cryptographic approach is more pragmatic

Provable Security

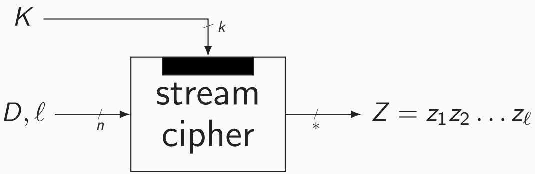
- Symmetric cryptographic schemes on number-theoretical problems exist, but are hardly ever practical
- Symmetric cryptographic approach is more pragmatic
- Primitives:
  - Considered secure if many people looked at it but nobody managed to break it
  - Some properties might still be provable (like: "certain attack approaches do not work")

Provable Security

- Symmetric cryptographic schemes on number-theoretical problems exist, but are hardly ever practical
- Symmetric cryptographic approach is more pragmatic
- Primitives:
  - Considered secure if many people looked at it but nobody managed to break it
  - Some properties might still be provable (like: "certain attack approaches do not work")
- Constructions:
  - Often come with a formal security proof
  - No unconditional security: based on assumption on the underlying primitive
  - Reductionist proof: breaking construction implies breaking primitive
  - Ideal model proof: assuming primitive is ideal, construction is secure

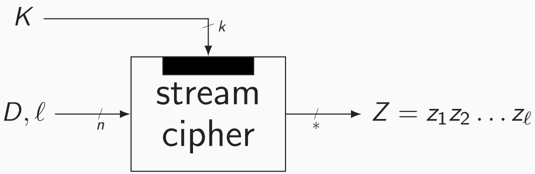
How to Model Security?

Modern Stream Ciphers



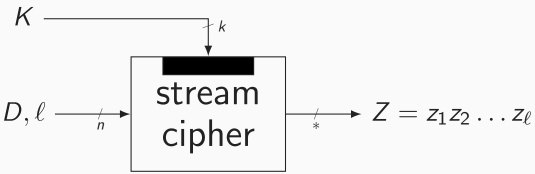
- Using key K, diversifier D, and length ℓ, keystream Z of length ℓ is generated

Modern Stream Ciphers



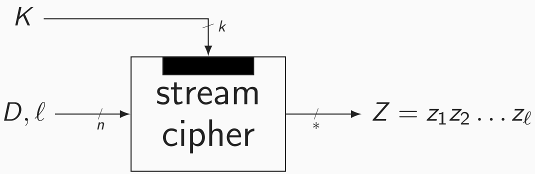
- Using key K, diversifier D, and length ℓ, keystream Z of length ℓ is generated
- The diversifier must be different for each message that is transmitted

Modern Stream Ciphers



- Using key K, diversifier D, and length ℓ, keystream Z of length ℓ is generated
- The diversifier must be different for each message that is transmitted
- Example: data streams, e.g., pay TV and telephone, often split data in relatively short, numbered, frames. One can use frame number as diversifier and encrypt using stream:

$$C_i = M_i \oplus F(K, i, |M_i|)$$



- Using key K, diversifier D, and length ℓ, keystream Z of length ℓ is generated
- The diversifier must be different for each message that is transmitted
- Example: data streams, e.g., pay TV and telephone, often split data in relatively short, numbered, frames. One can use frame number as diversifier and encrypt using stream:

$$C_i = M_i \oplus F(K, i, |M_i|)$$

When is a stream cipher strong enough?

Stream Cipher Security, Intuition (1/5)

SC_K
stream cipher

- Recall Kerckhoffs principle: security should be based on secrecy of K
- Consider attacker that learns some amount of input-output combinations of SC_K

Stream Cipher Security, Intuition (1/5)

SC_K
stream cipher

- Recall Kerckhoffs principle: security should be based on secrecy of K
- Consider attacker that learns some amount of input-output combinations of SC_K
- What should SC_K satisfy, intuitively?
  - It should be "hard" to recover the key, but is that all?

Stream Cipher Security, Intuition (1/5)

SC_K
stream cipher

- Recall Kerckhoffs principle: security should be based on secrecy of K
- Consider attacker that learns some amount of input-output combinations of SC_K
- What should SC_K satisfy, intuitively?
  - It should be "hard" to recover the key, but is that all?
  - If attacker ever sees $\ldots 1111111111 \ldots$ or $\ldots 0101010101 \ldots$, is that okay?
  - If attacker ever sees $\ldots 0101110101 \ldots$, is that okay?
  - $\ldots$

SC_K
stream cipher

- Recall Kerckhoffs principle: security should be based on secrecy of K
- Consider attacker that learns some amount of input-output combinations of SC_K
- What should SC_K satisfy, intuitively?
  - It should be "hard" to recover the key, but is that all?
  - If attacker ever sees $\ldots 1111111111 \ldots$ or $\ldots 0101010101 \ldots$, is that okay?
  - If attacker ever sees $\ldots 0101110101 \ldots$, is that okay?
  - $\ldots$
  - Intuitively, SC_K should not expose any irregularities
  - Its outputs should look completely random

Random Oracle

- A database of input-output tuples
- Initially empty

D	Z
$\ldots$	$\ldots$
$\ldots$	$\ldots$
$\ldots$	$\ldots$
$\ldots$	$\ldots$

Intermezzo: Random Oracle

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
  - If D is in the database,

D	Z
$\dots$	$\dots$
$\dots$	$\dots$
$\dots$	$\dots$
$\dots$	$\dots$

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database,

D	Z
$\ldots$	$\ldots$
$\ldots$	$\ldots$
$\ldots$	$\ldots$
$\ldots$	$\ldots$

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database,

D	Z
1100	101011101010101
...	...
...	...
...	...

Intermezzo: Random Oracle

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database,

D	Z
1100	101011101010101
1111010101101101	110101
...	...
...	...

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database,

D	Z
1100	101011101010101
1111010101101101	110101
001000011100	1010110101110101011
...	...

D	Z
1100	101011101010101
1111010101101101	110101
001000011100	1010110101110101011
...	...

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database, look at corresponding Z:
    - If $|Z| \geq \ell$:
    - If $|Z| < \ell$:

D	Z
1100	101011101010101
1111010101101101	110101
001000011100	1010110101110101011
...	...

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database, look at corresponding Z:
    - If $|Z| \geq \ell$: return first ℓ bits of Z
    - If $|Z| < \ell$:

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database, look at corresponding Z:
    - If $|Z| \geq \ell$: return first ℓ bits of Z
    - If $|Z| < \ell$:

D	Z
1100	101011101010101
1111010101101101	110101
001000011100	10101101011101010101011
...	...

D	Z
1100	101011101010101
1111010101101101	110101
001000011100	1010110101110101011
...	...

Random Oracle

- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database, look at corresponding Z:
    - If $|Z| \geq \ell$: return first ℓ bits of Z
    - If $|Z| < \ell$: generate $\ell - |Z|$ random bits Z', append Z' to Z, return $Z \| Z'$

D	Z
1100	101011101010101
1111010101101101	1101011101111101101
001000011100	1010110101110101011
...	...

Random Oracle

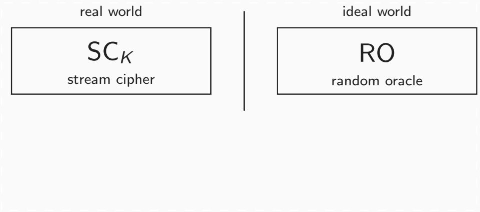
- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database, look at corresponding Z:
    - If $|Z| \geq \ell$: return first ℓ bits of Z
    - If $|Z| < \ell$: generate $\ell - |Z|$ random bits Z', append Z' to Z, return $Z \| Z'$

D	Z
1100	101011101010101
1111010101101101	1101011101111101101
001000011100	10101101011101010101011
...	...

Random Oracle

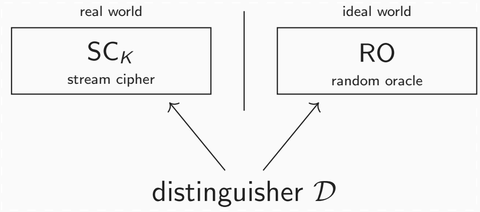
- A database of input-output tuples
- Initially empty
- New query (D, ℓ):
  - If D is not in the database:
    - generate ℓ random bits Z
    - add (D, Z) to the list
    - return Z
  - If D is in the database, look at corresponding Z:
    - If $|Z| \geq \ell$: return first ℓ bits of Z
    - If $|Z| < \ell$: generate $\ell - |Z|$ random bits Z', append Z' to Z, return $Z \| Z'$
    - update (D, Z) in the list

Stream Cipher Security, Intuition (2/5)

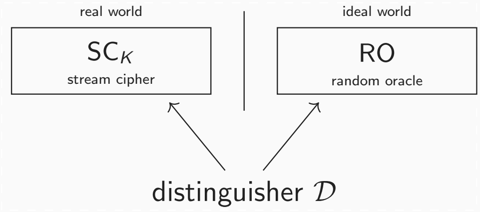


- We thus want to "compare" SC_K with a random oracle RO

Stream Cipher Security, Intuition (2/5)

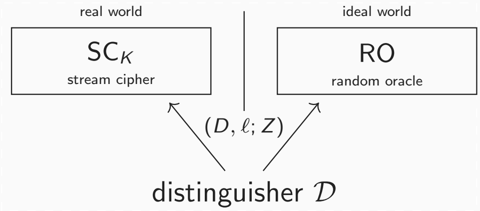


- We thus want to "compare" SC_K with a random oracle RO
- We model a distinguisher $\mathcal{D}$ that is given oracle access to either of the worlds



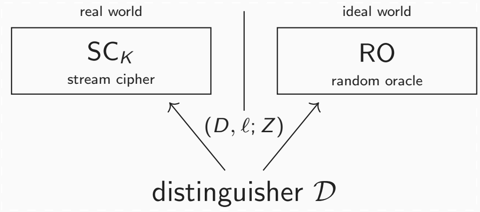
- We thus want to "compare" SC_K with a random oracle RO
- We model a distinguisher $\mathcal{D}$ that is given oracle access to either of the worlds
  - We toss a coin:
    - head: $\mathcal{D}$ is given oracle access to SC_K
    - tail: $\mathcal{D}$ is given oracle access to RO
  - $\mathcal{D}$ does a priori not know which oracle it is given access to

Stream Cipher Security, Intuition (2/5)



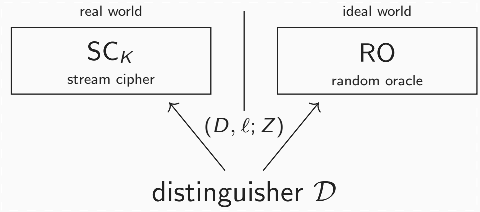
- We thus want to "compare" SC_K with a random oracle RO
- We model a distinguisher $\mathcal{D}$ that is given oracle access to either of the worlds
    - We toss a coin:
        - head: $\mathcal{D}$ is given oracle access to SC_K
        - tail: $\mathcal{D}$ is given oracle access to RO
    - $\mathcal{D}$ does a priori not know which oracle it is given access to
    - $\mathcal{D}$ can now make queries (D, ℓ) to receive Z

Stream Cipher Security, Intuition (2/5)

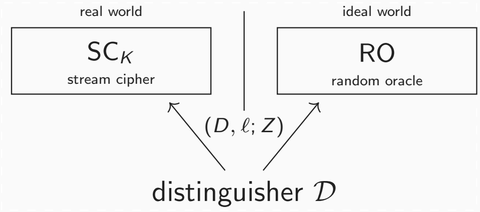


- We thus want to "compare" SC_K with a random oracle RO
- We model a distinguisher $\mathcal{D}$ that is given oracle access to either of the worlds
  - We toss a coin:
    - head: $\mathcal{D}$ is given oracle access to SC_K
    - tail: $\mathcal{D}$ is given oracle access to RO
  - $\mathcal{D}$ does a priori not know which oracle it is given access to
  - $\mathcal{D}$ can now make queries (D, ℓ) to receive Z
  - At the end, $\mathcal{D}$ has to guess the outcome of the coin toss (head/tail)

Stream Cipher Security, Intuition (3/5)



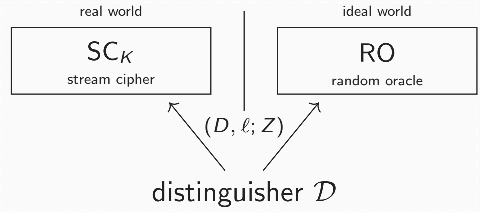
- Denote $\mathcal{D}$'s success probability in correctly guessing head/tail by $\mathbf{Pr}(\text{success})$



- Denote $\mathcal{D}$'s success probability in correctly guessing head/tail by $\mathbf{Pr}\left(\text{success}\right)$
- $\mathcal{D}$ can always guess and succeeds with probability $\geq 1/2$, so we scale the probability to $\mathcal{D}$'s advantage:

$$\mathbf{Adv}(\mathcal{D}) = 2 \cdot \mathbf{Pr}\left(\text{success}\right) - 1$$

Stream Cipher Security, Intuition (3/5)



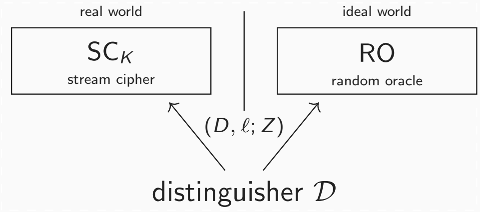
- Denote $\mathcal{D}$'s success probability in correctly guessing head/tail by $\mathbf{Pr}\,(\text{success})$
- $\mathcal{D}$ can always guess and succeeds with probability $\geq 1/2$, so we scale the probability to $\mathcal{D}$'s advantage:

$$\mathbf{Adv}(\mathcal{D}) = 2 \cdot \mathbf{Pr}\,(\text{success}) - 1$$

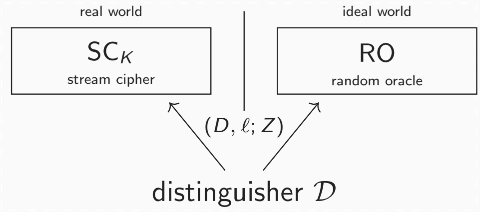
- This turns out to be equal to (see Section 4.4 of "Intro2Crypto-symmetric.pdf")

$$\mathbf{Adv}(\mathcal{D}) = \mathbf{Pr}\left(\mathcal{D}^{\mathsf{SC}_K} \text{ returns head}\right) - \mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} \text{ returns head}\right)$$

Stream Cipher Security, Intuition (4/5)



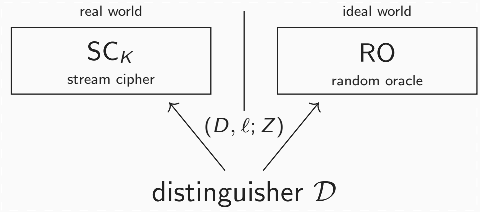
- Recall: distinguisher is limited by certain constraints
  - data complexity: amount of observed data (limited by use case)
  - computation complexity: amount of computation (limited by budget)
- How do these constraints relate to the security model?



- Recall: distinguisher is limited by certain constraints
  - data complexity: amount of observed data (limited by use case)
  - computation complexity: amount of computation (limited by budget)
- How do these constraints relate to the security model?

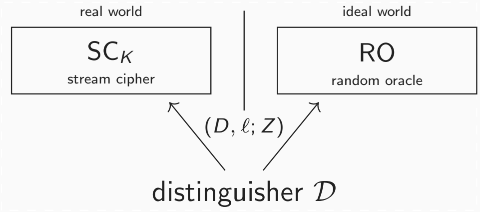
- Data (or online) complexity q: total cost of queries $\mathcal{D}$ can make

Stream Cipher Security, Intuition (4/5)



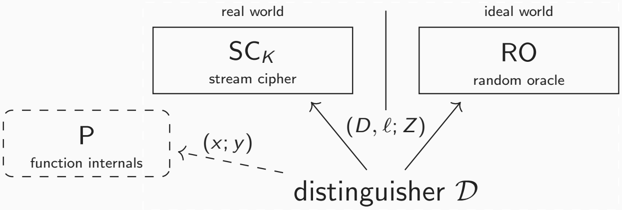
- Recall: distinguisher is limited by certain constraints
  - data complexity: amount of observed data (limited by use case)
  - computation complexity: amount of computation (limited by budget)
- How do these constraints relate to the security model?

- Data (or online) complexity q: total cost of queries $\mathcal{D}$ can make
- Computation (or time) complexity t: everything that $\mathcal{D}$ can do "on its own"



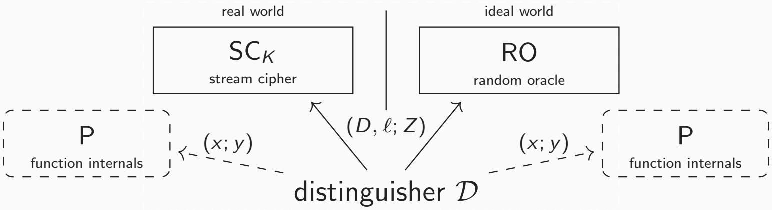
- Computation (or time) complexity t: everything that $\mathcal{D}$ can do "on its own"
  - SC (without key input) is a public algorithm
  - $\mathcal{D}$ can evaluate it offline
  - For instance, it can try evaluate $SC_{K'}$ for different keys K'

Stream Cipher Security, Intuition (5/5)



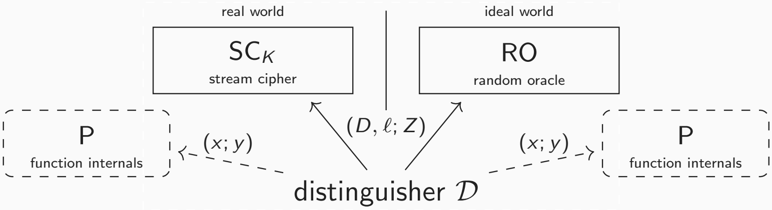
- Computation (or time) complexity t: everything that $\mathcal{D}$ can do "on its own"
  - SC (without key input) is a public algorithm
  - $\mathcal{D}$ can evaluate it offline
  - For instance, it can try evaluate $\mathsf{SC}_{K'}$ for different keys K'
  - Even stronger: $\mathcal{D}$ can evaluate individual internal parts of SC offline

Stream Cipher Security, Intuition (5/5)



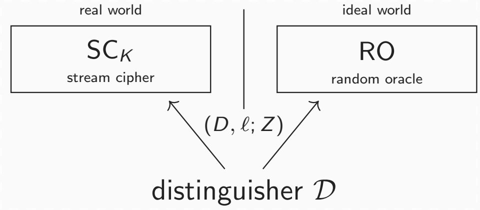
- Computation (or time) complexity t: everything that $\mathcal{D}$ can do "on its own"
  - SC (without key input) is a public algorithm
  - $\mathcal{D}$ can evaluate it offline
  - For instance, it can try evaluate $SC_{K'}$ for different keys K'
  - Even stronger: $\mathcal{D}$ can evaluate individual internal parts of SC offline
  - It can do so regardless of the oracle it is communicating with

Stream Cipher Security, Intuition (5/5)

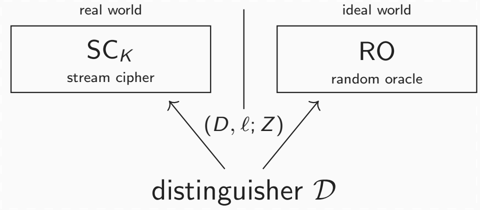


- Computation (or time) complexity t: everything that $\mathcal{D}$ can do "on its own"
  - SC (without key input) is a public algorithm
  - $\mathcal{D}$ can evaluate it offline
  - For instance, it can try evaluate $\mathsf{SC}_{K'}$ for different keys K'
  - Even stronger: $\mathcal{D}$ can evaluate individual internal parts of SC offline
  - It can do so regardless of the oracle it is communicating with
  - Offline access to these internals is, however, often left implicit

Stream Cipher Security, Formal (1/2)

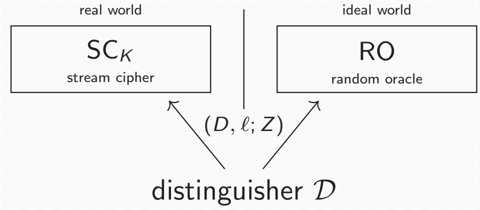


- Two oracles: SC_K (for secret key K) and RO (secret)



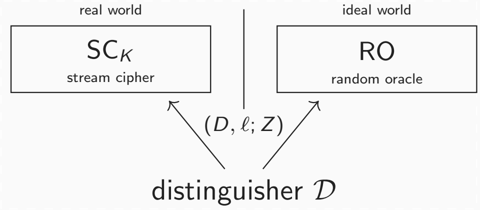
- Two oracles: SC_K (for secret key K) and RO (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these

Stream Cipher Security, Formal (1/2)



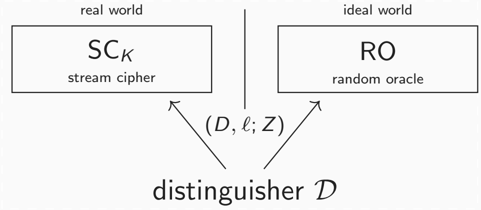
- Two oracles: SC_K (for secret key K) and RO (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with

Stream Cipher Security, Formal (1/2)



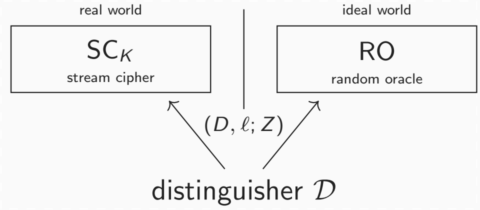
- Two oracles: SC_K (for secret key K) and RO (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(\mathsf{SC}_K \,;\, \mathsf{RO}\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{SC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} = 1\right)\right|$$



- Its advantage is defined as:

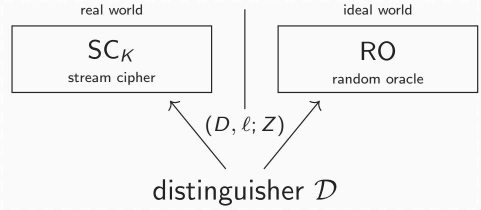
$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(\mathsf{SC}_K \, ; \, \mathsf{RO}\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{SC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} = 1\right)\right|$$



- Its advantage is defined as:

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(\mathsf{SC}_K \;;\; \mathsf{RO}\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{SC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} = 1\right)\right|$$

- $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$: supremal advantage over any distinguisher with complexity q, t



- Its advantage is defined as:

$$\mathbf{Adv}_{\mathsf{SC}}^{\mathrm{prf}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(\mathsf{SC}_K \;;\; \mathsf{RO}\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{\mathsf{SC}_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{\mathsf{RO}} = 1\right)\right|$$

- $\mathbf{Adv}_{\mathsf{SC}}^{\mathrm{prf}}(q, t)$: supremal advantage over any distinguisher with complexity q, t
  - More complexity parameters may apply, e.g., total length, different complexity bounds for different oracles, ...
  - In addition, t is sometimes left implicit if not needed for a security proof

Stream Cipher Security, Implication

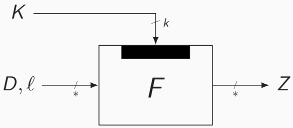
- A bound $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$ implies that
  - no key recovery attack succeeds with advantage higher than $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$
  - no bias in keystream can be exploited with advantage higher than $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$
  - ...
  - no meaningful attack can be mounted with advantage higher than $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$

Stream Cipher Security, Implication

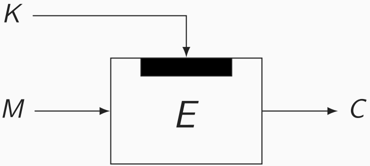
- A bound $\mathbf{Adv}_{\mathsf{SC}}^{\mathrm{prf}}(q,t)$ implies that
  - no key recovery attack succeeds with advantage higher than $\mathbf{Adv}_{\mathsf{SC}}^{\mathrm{prf}}(q,t)$
  - no bias in keystream can be exploited with advantage higher than $\mathbf{Adv}_{\mathsf{SC}}^{\mathrm{prf}}(q,t)$
  - ...
  - no meaningful attack can be mounted with advantage higher than $\mathbf{Adv}_{\mathsf{SC}}^{\mathrm{prf}}(q,t)$
- Bound on the advantage can serve two purposes:
  - Security claim for a concrete design
  - A proven security bound assuming security of an underlying building block

Stream Cipher Security, Implication

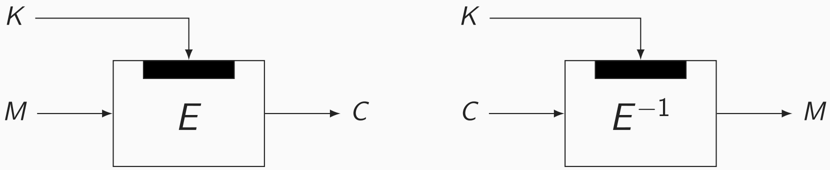
- A bound $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$ implies that
  - no key recovery attack succeeds with advantage higher than $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$
  - no bias in keystream can be exploited with advantage higher than $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$
  - ...
  - no meaningful attack can be mounted with advantage higher than $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(q, t)$
- Bound on the advantage can serve two purposes:
  - Security claim for a concrete design
  - A proven security bound assuming security of an underlying building block
- Security definition of pseudorandom functions (PRF) is in fact more general: it applies to functions with possibly arbitrary length inputs and outputs



Block Ciphers

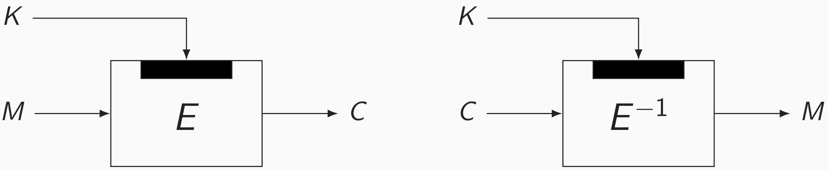


- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size

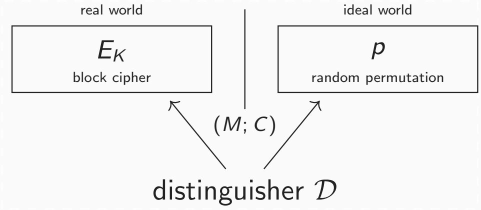


- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size
- For fixed key, E_K is invertible and the inverse is denoted as E_K^{-1}

Block Ciphers

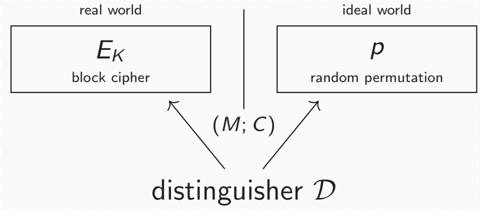


- Using key K, message M is bijectively transformed to ciphertext C
- Key, plaintext, and ciphertext are typically of fixed size
- For fixed key, E_K is invertible and the inverse is denoted as E_K^{-1}
- A good block cipher should behave like a random permutation



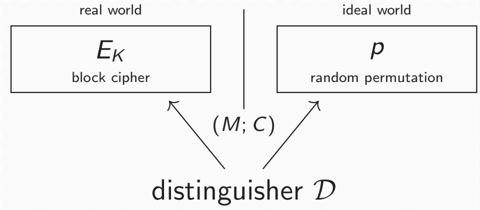
- Two oracles: E_K (for secret key K) and p (secret)

Block Cipher Security



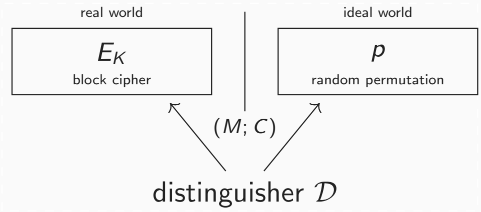
- Two oracles: E_K (for secret key K) and p (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these

Block Cipher Security



- Two oracles: E_K (for secret key K) and p (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with

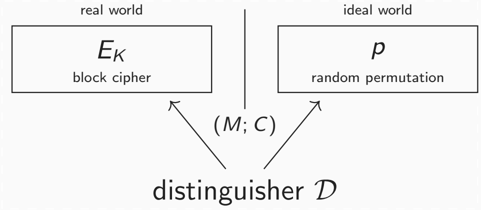
Block Cipher Security



- Two oracles: E_K (for secret key K) and p (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

$$\mathbf{Adv}_E^{\mathrm{prp}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(E_K \; ; \; p\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{E_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{p} = 1\right)\right|$$

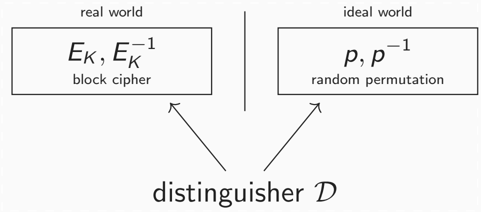
Block Cipher Security



- Two oracles: E_K (for secret key K) and p (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

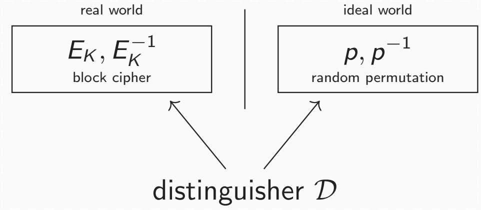
$$\mathbf{Adv}_E^{\mathrm{prp}}(\mathcal{D}) = \Delta_{\mathcal{D}}(E_K\,;\,p) = \left|\mathbf{Pr}\left(\mathcal{D}^{E_K} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{p} = 1\right)\right|$$

- $\mathbf{Adv}_E^{\mathrm{prp}}(q, t)$: supremal advantage over any $\mathcal{A}$ with query/time complexity q/t

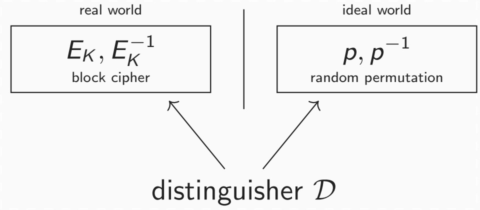


- Two oracles: (E_K, E_K^{-1}) (for secret key K) and (p, p^{-1}) (secret)

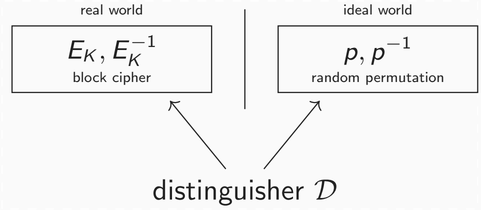
Strong Block Cipher Security



- Two oracles: (E_K, E_K^{-1}) (for secret key K) and (p, p^{-1}) (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these



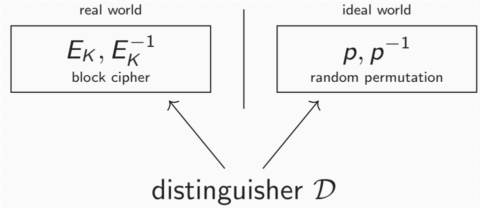
- Two oracles: (E_K, E_K^{-1}) (for secret key K) and (p, p^{-1}) (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with



- Two oracles: (E_K, E_K^{-1}) (for secret key K) and (p, p^{-1}) (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

$$\mathbf{Adv}_E^{\mathrm{sprp}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(E_K, E_K^{-1}\ ;\ p, p^{-1}\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{E_K, E_K^{-1}} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{p, p^{-1}} = 1\right)\right|$$

Strong Block Cipher Security

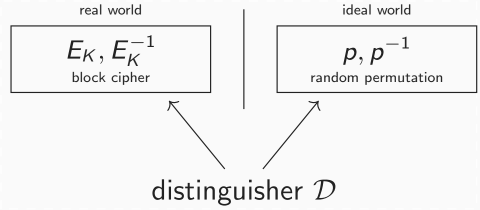


- Two oracles: (E_K, E_K^{-1}) (for secret key K) and (p, p^{-1}) (secret)
- Distinguisher $\mathcal{D}$ has query access to one of these
- $\mathcal{D}$ tries to determine which oracle it communicates with
- Its advantage is defined as:

$$\mathbf{Adv}_E^{\mathrm{sprp}}(\mathcal{D}) = \Delta_{\mathcal{D}}\left(E_K, E_K^{-1}\ ;\ p, p^{-1}\right) = \left|\mathbf{Pr}\left(\mathcal{D}^{E_K, E_K^{-1}} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{p, p^{-1}} = 1\right)\right|$$

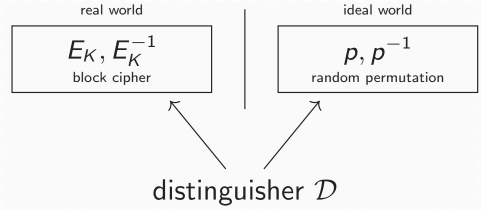
- $\mathbf{Adv}_E^{\mathrm{sprp}}(q, t)$: supremal advantage over any $\mathcal{A}$ with query/time complexity q/t

Block Cipher Security: How to Model Key Recovery?



- Suppose $\mathcal{D}$ has $q \geq 1$ query and t time

Block Cipher Security: How to Model Key Recovery?



- Suppose $\mathcal{D}$ has $q \geq 1$ query and t time
- It can mount the following attack:
  - Make 1 construction query $(0; \mathcal{O}(0))$
  - Make t offline key attempts $E_{L_i}(0)$
  - If $E_{L_i}(0) = \mathcal{O}(0)$ for some i, key recovery very likely
- For this distinguisher (simplified, ignoring false positives): $\mathbf{Adv}_E^{\mathrm{sprp}}(\mathcal{D}) \approx t/2^k$
- Supremized: $\mathbf{Adv}_E^{\mathrm{sprp}}(q, t) \geq t/2^k$

AES

- Block cipher with block and key lengths $\in \{128, 160, 192, 224, 256\}$
  - Set of 25 block ciphers
  - AES limits block length to 128 and key length to multiples of 64
- We only consider AES in this course

AES

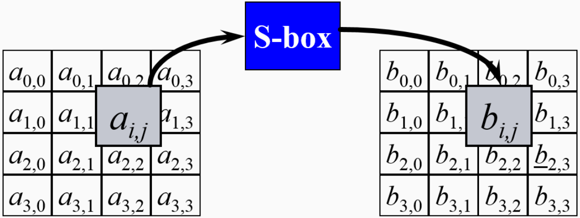
- Block cipher with block and key lengths $\in \{128, 160, 192, 224, 256\}$
  - Set of 25 block ciphers
  - AES limits block length to 128 and key length to multiples of 64
- We only consider AES in this course
- Iteration of a round function with following properties:
  - 4 layers: nonlinear, shuffling, mixing and round key addition
  - All rounds are identical ...
  - ... except for the round keys
  - ... and omission of mixing layer in last round
  - Parallel and symmetric

AES

- Block cipher with block and key lengths $\in \{128, 160, 192, 224, 256\}$
  - Set of 25 block ciphers
  - AES limits block length to 128 and key length to multiples of 64
- We only consider AES in this course
- Iteration of a round function with following properties:
  - 4 layers: nonlinear, shuffling, mixing and round key addition
  - All rounds are identical ...
  - ... except for the round keys
  - ... and omission of mixing layer in last round
  - Parallel and symmetric
- Key schedule
  - Expansion of cipher key to round key sequence
  - Recursive procedure that can be done in-place

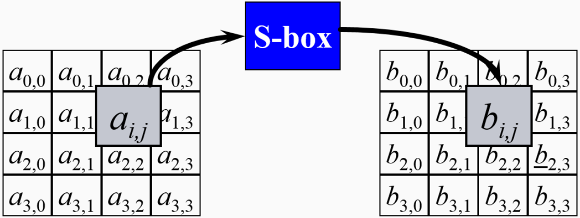
- Block cipher with block and key lengths $\in \{128, 160, 192, 224, 256\}$
  - Set of 25 block ciphers
  - AES limits block length to 128 and key length to multiples of 64
- We only consider AES in this course
- Iteration of a round function with following properties:
  - 4 layers: nonlinear, shuffling, mixing and round key addition
  - All rounds are identical . . .
  - . . . except for the round keys
  - . . . and omission of mixing layer in last round
  - Parallel and symmetric
- Key schedule
  - Expansion of cipher key to round key sequence
  - Recursive procedure that can be done in-place
- Manipulates bytes rather than bits

The Non-Linear Layer: SubBytes



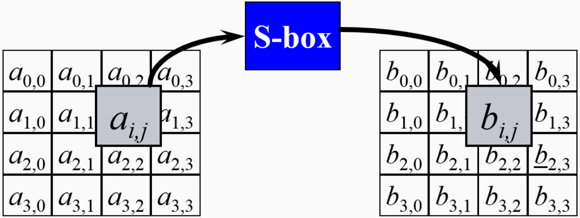
- The same invertible S-box applied to all bytes of the state

The Non-Linear Layer: SubBytes



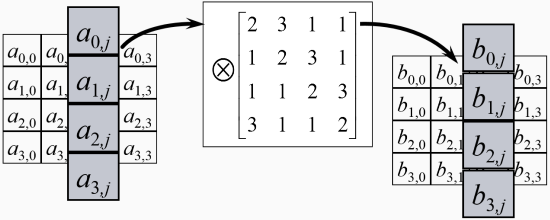
- The same invertible S-box applied to all bytes of the state
- Assembled from building blocks that were proposed and analyzed in cryptographic literature

The Non-Linear Layer: SubBytes



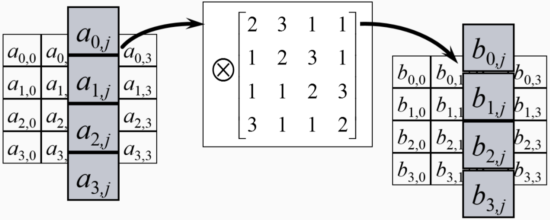
- The same invertible S-box applied to all bytes of the state
- Assembled from building blocks that were proposed and analyzed in cryptographic literature
- Criteria:
  - Offer resistance against DC, LC and *algebraic* attacks . . .
  - . . . when combined with the other layers

The Mixing Layer: MixColumns



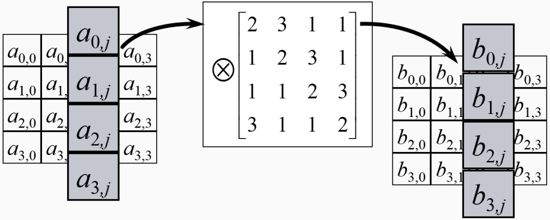
- Same invertible mapping applied to all 4 columns

The Mixing Layer: MixColumns



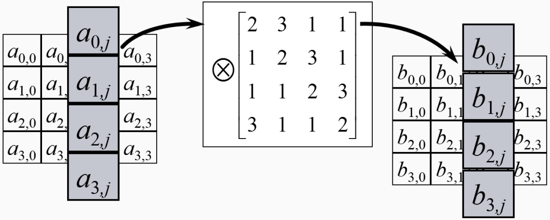
- Same invertible mapping applied to all 4 columns
- Multiplication by a 4×4 circulant matrix in $\mathrm{GF}(2^8)$
    - Difference in 1 input byte propagates to 4 output bytes

The Mixing Layer: MixColumns



- Same invertible mapping applied to all 4 columns
- Multiplication by a 4×4 circulant matrix in $\mathrm{GF}(2^8)$
  - Difference in 1 input byte propagates to 4 output bytes
  - Difference in 2 input bytes propagates to 3 output bytes

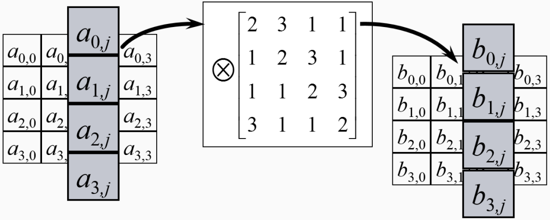
The Mixing Layer: MixColumns



- Same invertible mapping applied to all 4 columns
- Multiplication by a 4×4 circulant matrix in $\mathrm{GF}(2^8)$
  - Difference in 1 input byte propagates to 4 output bytes
  - Difference in 2 input bytes propagates to 3 output bytes
  - Difference in 3 input bytes propagates to 2 output bytes

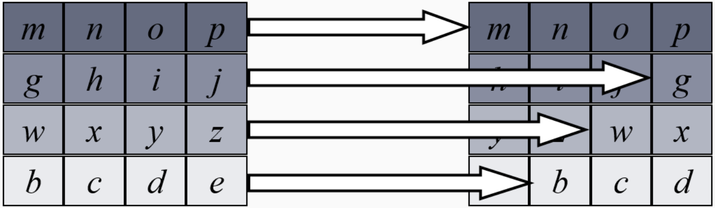
Slide credit: Joan Daemen

The Mixing Layer: MixColumns



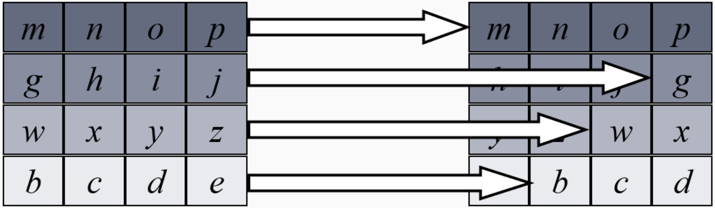
- Same invertible mapping applied to all 4 columns
- Multiplication by a 4×4 circulant matrix in $\mathrm{GF}(2^8)$
  - Difference in 1 input byte propagates to 4 output bytes
  - Difference in 2 input bytes propagates to 3 output bytes
  - Difference in 3 input bytes propagates to 2 output bytes
  - Difference in 4 input bytes propagates to 1 output byte

The Shuffling Layer: ShiftRows



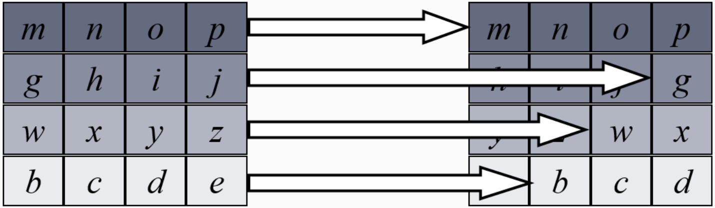
- Each row is shifted by a different amount

The Shuffling Layer: ShiftRows



- Each row is shifted by a different amount
- Different shift offsets for higher block lengths

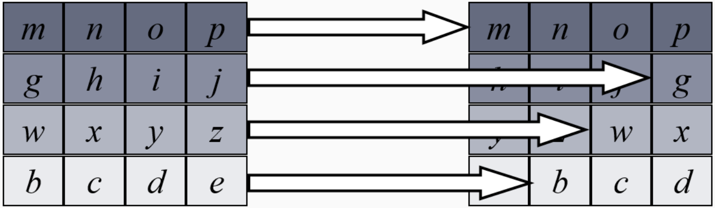
The Shuffling Layer: ShiftRows



- Each row is shifted by a different amount
- Different shift offsets for higher block lengths
- Moves bytes in a given column to 4 different columns

Slide credit: Joan Daemen

The Shuffling Layer: ShiftRows



- Each row is shifted by a different amount
- Different shift offsets for higher block lengths
- Moves bytes in a given column to 4 different columns
- Combined with MixColumns and SubBytes this gives fast diffusion

Round Key Addition: AddRoundKey

$$\begin{array}{|c|c|c|c|}\hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} + \begin{array}{|c|c|c|c|}\hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

- Round key is computed from the cipher key K

Key Schedule: Example with 192-bit Key K

k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}	$\cdots$

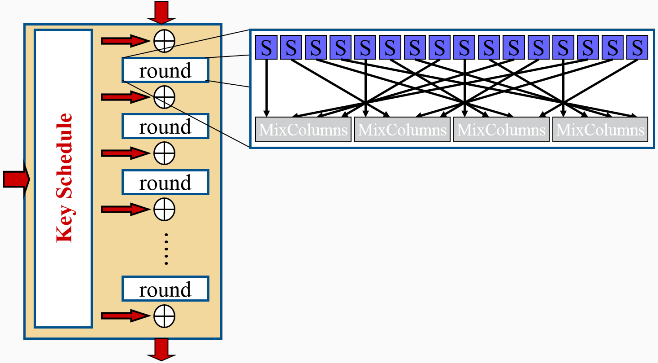
Round key 0	Round key 1	Round key 2	$\cdots$

- Expansion: put K in 1st columns and compute others recursively:

$$k_{6n} = k_{6n-6} \oplus f(k_{6n-1})$$
$$k_i = k_{i-6} \oplus k_{i-1}, \quad i \neq 6n$$

  with f: 4 parallel AES S-boxes followed by 1-byte cyclic shift
- Selection: round key i is columns $4i$ to $4i+3$

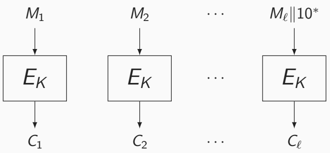
AES: Summary



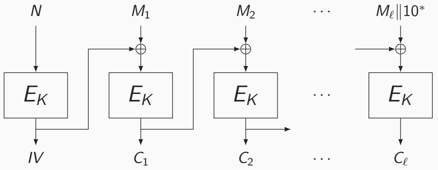
- 10 rounds for 128-bit key, 12 for 192-bit key and 14 for 256-bit key
- Last round has no MixColumns so that inverse is similar to cipher

Block Cipher Based Encryption Modes

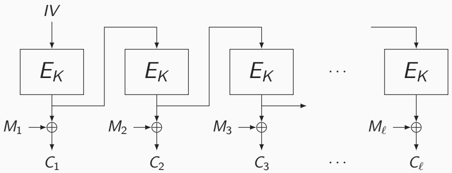
Block Cipher Encryption Modes



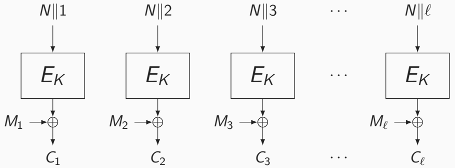
ECB



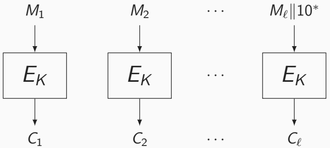
CBC



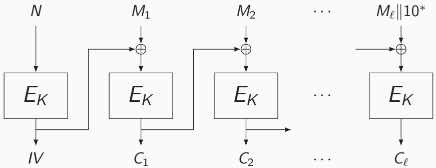
OFB



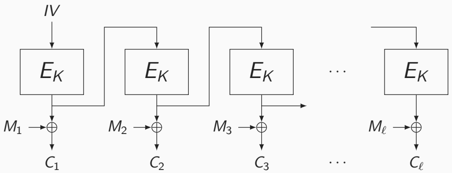
CTR



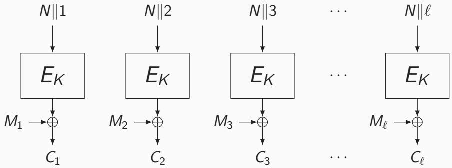
ECB



CBC



OFB



CTR

Open question: advantages/disadvantages?

Overview

	ECB	CBC	OFB	CTR
parallel encryption	✓	—	—	✓
parallel decryption	✓	✓	—	✓
inverse free	—	—	✓	✓
absence of message expansion	—	—	✓	✓
tolerant to bit flips in $C \to P$	—	—	✓	✓
graceful degradation if nonce violation	n/a	✓	—	—

Conclusion

Conclusion

- Cryptographic functions: often expected to behave like random oracles

Conclusion

- Cryptographic functions: often expected to behave like random oracles
- Designing fixed-length primitives that behave like random functions is harder than one might think

Conclusion

- Cryptographic functions: often expected to behave like random oracles
- Designing fixed-length primitives that behave like random functions is harder than one might think
- Easier to design fixed-length primitives that behave like random permutations

Conclusion

- Cryptographic functions: often expected to behave like random oracles
- Designing fixed-length primitives that behave like random functions is harder than one might think
- Easier to design fixed-length primitives that behave like random permutations
- At "Introduction to Cryptography", you learned about some symmetric cryptographic designs

Conclusion

- Cryptographic functions: often expected to behave like random oracles
- Designing fixed-length primitives that behave like random functions is harder than one might think
- Easier to design fixed-length primitives that behave like random permutations
- At "Introduction to Cryptography", you learned about some symmetric cryptographic designs

Next Lectures

- Advanced techniques on how to argue security
- More involved functions such as authenticated encryption
- Standardization efforts (NIST, ISO, CFRG, PKCS)