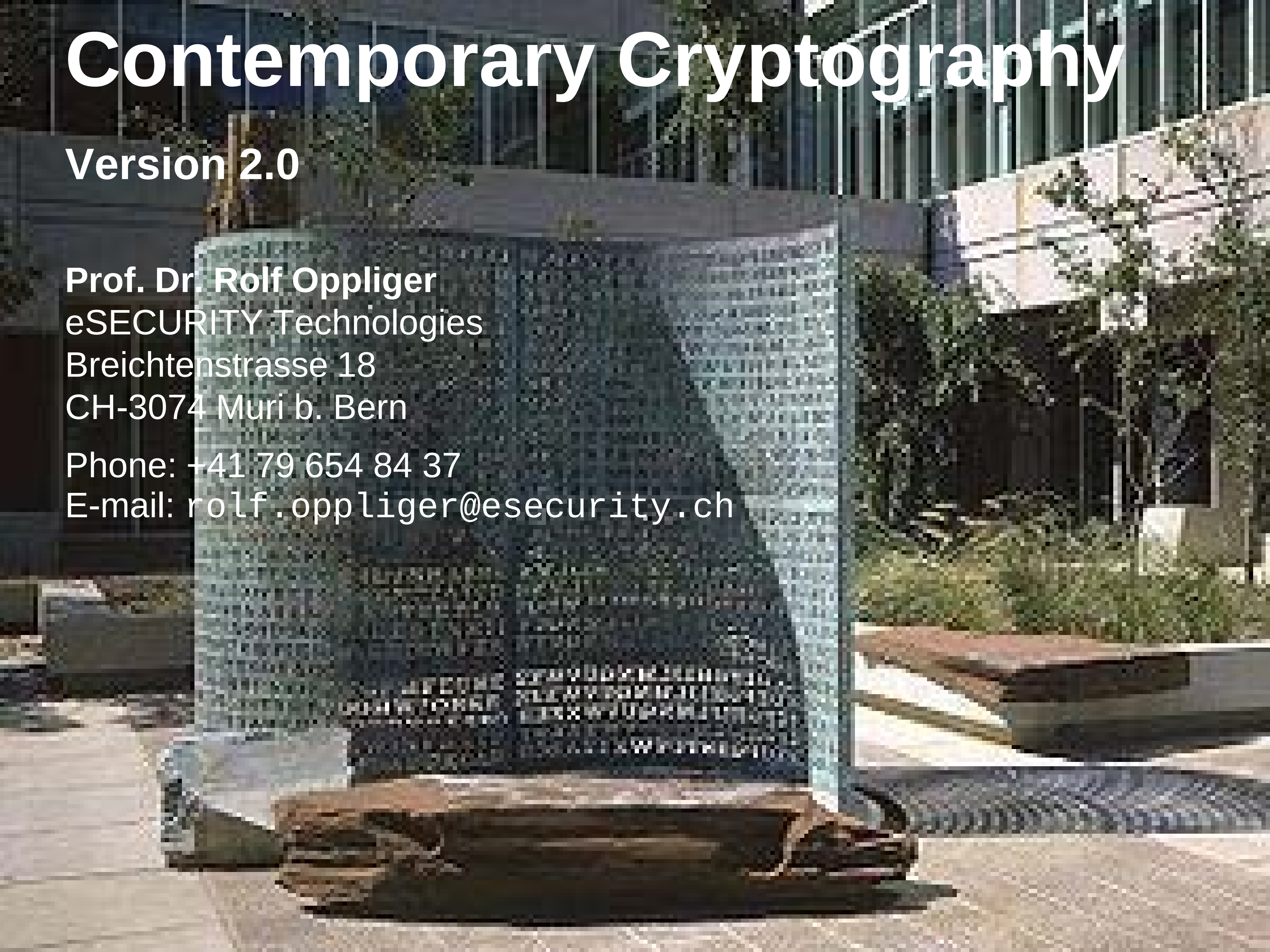


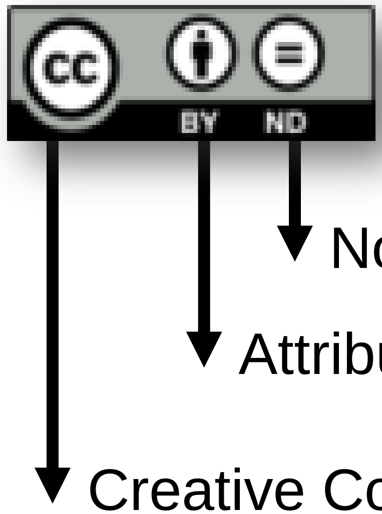
Contemporary Cryptography

Version 2.0

Prof. Dr. Rolf Oppliger
eSECURITY Technologies
Breichtenstrasse 18
CH-3074 Muri b. Bern

Phone: +41 79 654 84 37
E-mail: `rolf.oppliger@esecurity.ch`

Terms of Use (Copyright)



Instructor

Professional activties

- eSECURITY Technologies Rolf Oppliger (founder and owner)
- Swiss Federal Stratgey Unit for Information Technology (scientific employee)
- University of Zurich (adjunct professor)
- Artech House (author and series editor for information security and privacy)

→ http://www.esecurity.ch/bio.html

Module Overview

Module 1

Introduction and Overview

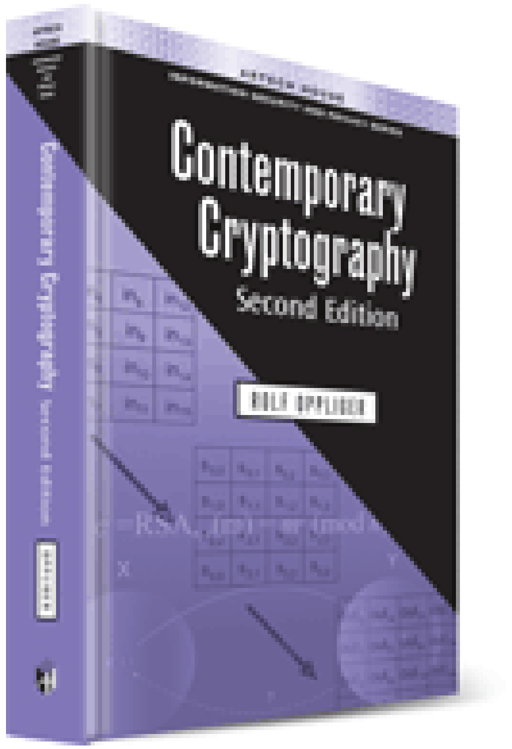
Module 2

Secret Key Cryptography

Module 3

Module 4

Reference Book



© Artech House (2011)
ISBN 978-1-60807-145-6

→ http://books.esecurity.ch/cryptography2e.html

Alternative Books

- Menezes, A.J., van Oorschot, P.C., and S.A. Vanstone, **Handbook of Applied Cryptography**, ISBN 0849385237, CRC Press, Boca Raton, FL, 1996 (→ http://www.cacr.math.uwaterloo.ca/hac)
- Stinson, D., **Cryptography: Theory and Practice**, 3rd Edition, ISBN 1584885084, Chapman & Hall/ CRC, Boca Raton, FL, 2005
- Paar, C., and J. Pelzl, **Understanding Cryptography: A Textbook for Students and Practitioners**, ISBN 3642041000, Springer, 2009
- Mao, W., **Modern Cryptography: Theory and Practice**, ISBN 0130669431, Prentice Hall PTR, Upper Saddle River, NJ, 2003
- Delfs, H., and H. Knebl, **Introduction to Cryptography: Principles and Applications**, 2nd Edition, ISBN 3540492437, Springer, 2007
- Dent, A.W., and C.J. Mitchell, **User's Guide to Cryptography and Standards**, ISBN 1580535305, Artech House, Norwood, MA, 2004

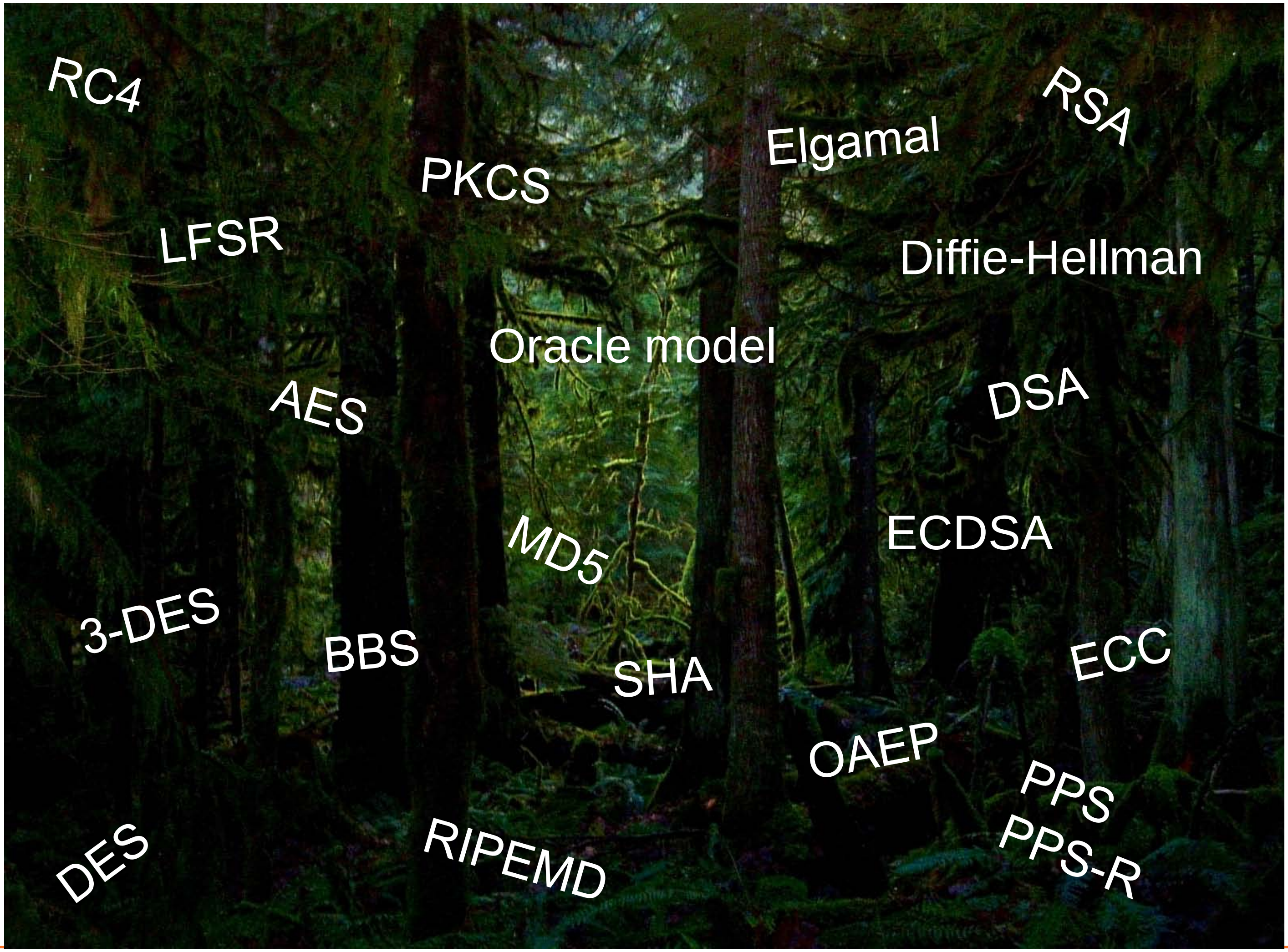
Preliminary Remarks

Necessity is the mother of invention,
and computer networks are the mother of modern cryptography

- Ronald L. Rivest (1997*)

* In: CRYPTOGRAPHY AS DUCT TAPE, 6/12/1997 → http://people.csail.mit.edu/rivest/Ducttape.txt

Preliminary Remarks

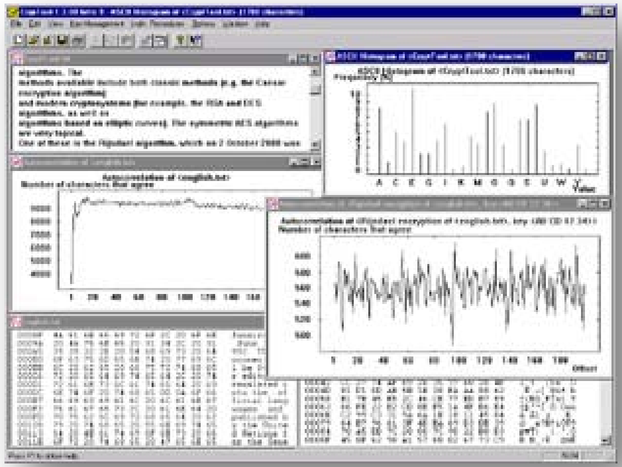


Preliminary Remarks

- Cryptography is an enabling technology (or tool) to secure the information infrastructure(s) we
  - build,
  - use, and
  - count on in daily life
- Computer scientists, electrical engineers, and applied mathematicians should care about (and be educated in) the principles and applications of cryptography

Preliminary Remarks

- This seminar addresses only the materials that are available in public, i.e., no rumors or unverified claims
- There are tools to experiment with cryptographic systems
- For example, CrypTool is publicly and freely available (→ http://www.cryptool.org)
- A few exercises are included in the seminar (optional)



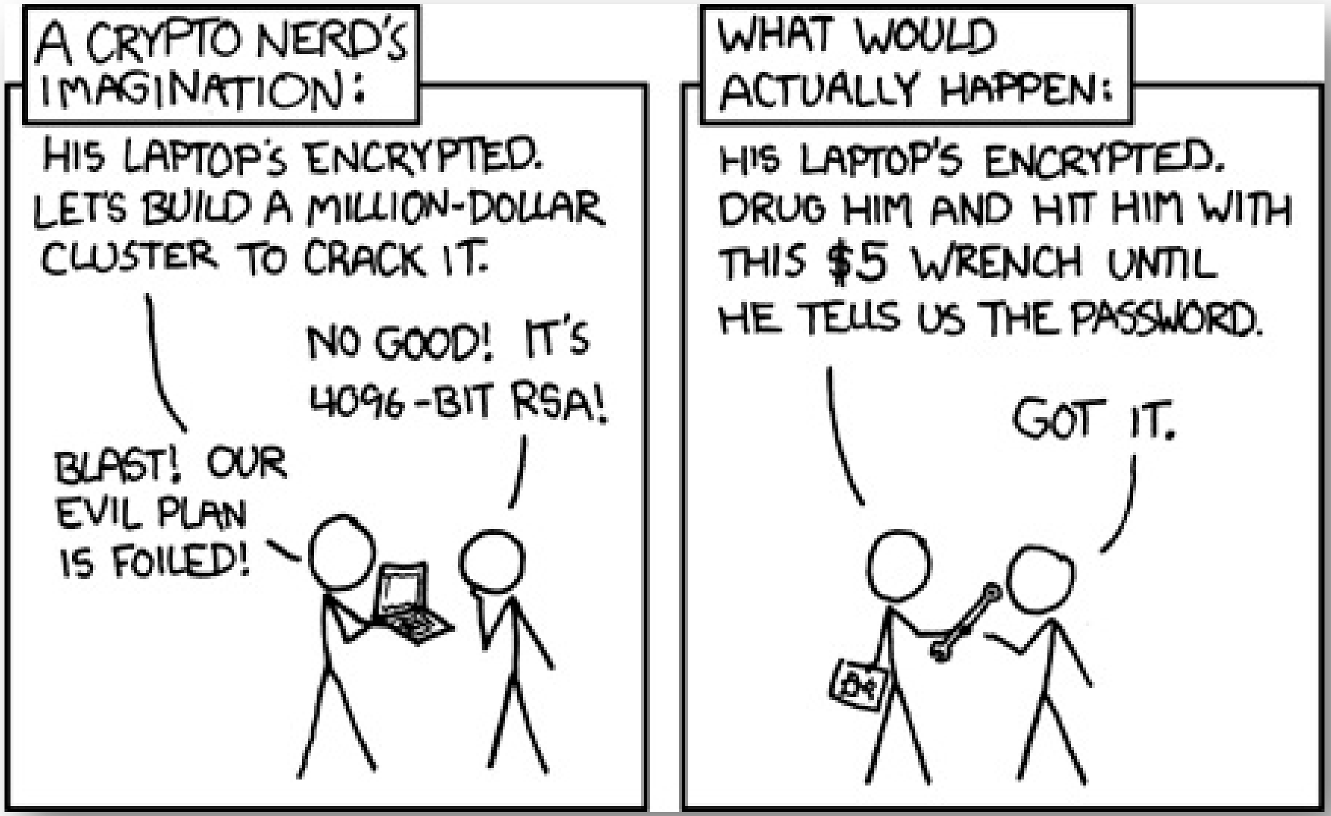
Preliminary Remarks

- The slides are relatively simple, down-to-earth, and not visually stimulating
- Mathematical fundamentals are not addressed
- Alice, Bob, Carol, Dave, Eve, and the rest of the gang are posted as missing

OH ALICE... YOU'RE THE ONE FOR ME
BUT BOB... IN A QUANTUM WORLD HOW CAN WE BE SURE?
ψ^+ or ψ^- ?

Preliminary Remarks

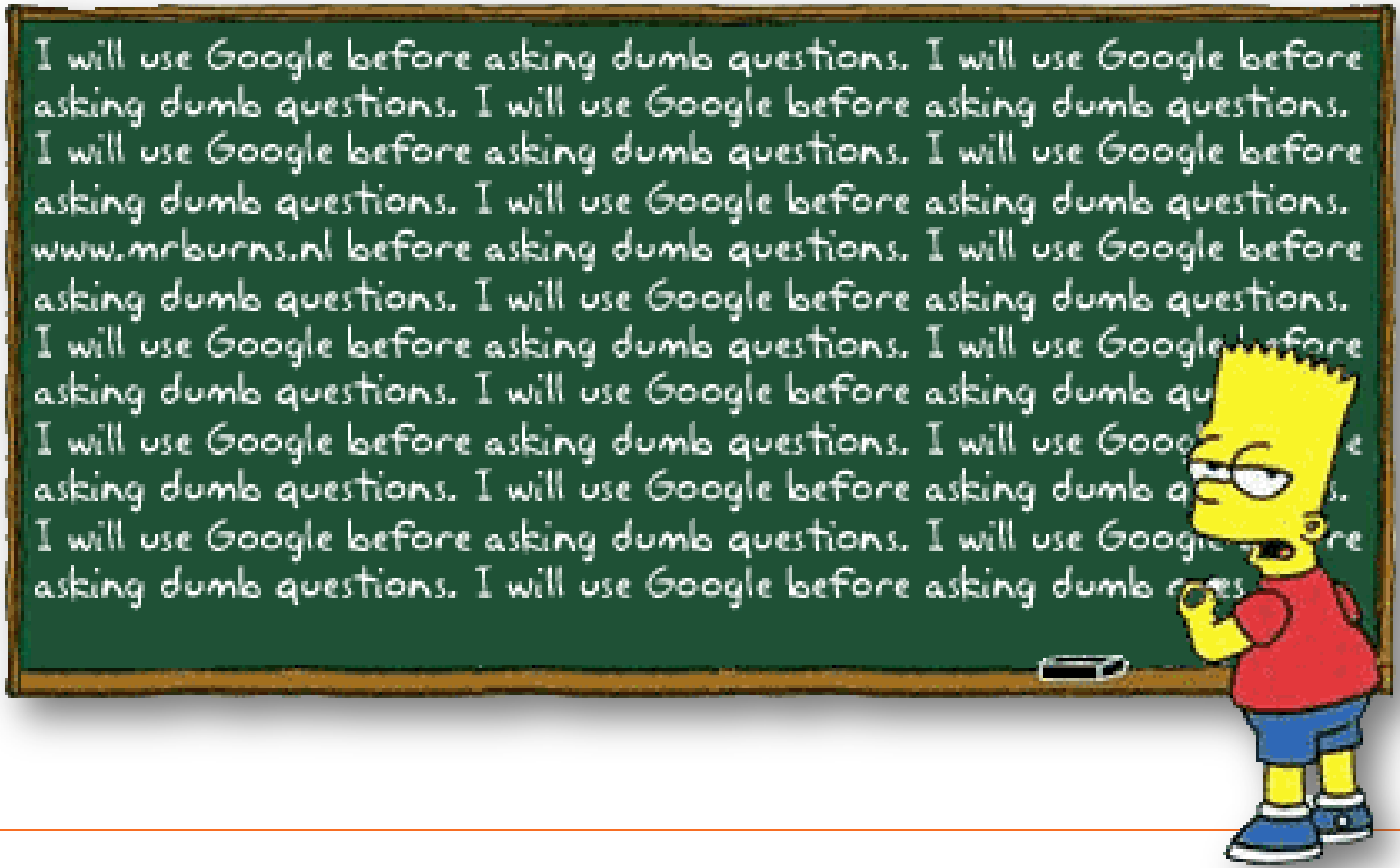
- There are inherent limitations of cryptography ...


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

http://xkcd.com/538/

Questions

- Please, interrupt and ask questions at will!


I will use Google before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb questions.
I will use Google before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb questions.
www.mrburns.nl before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb questions.
I will use Google before asking dumb questions. I will use Google before
asking dumb questions. I will use Google before asking dumb qu
I will use Google before asking dumb questions. I will use Goog
asking dumb questions. I will use Google before asking dumb q
I will use Google before asking dumb questions. I will use Goog
asking dumb questions. I will use Google before asking dumb ques

Module 1

Introduction and Overview

1 Introduction

Introduction

1.1 Cryptology

- The term cryptology is derived from two Greek words
  - kryptós ≈ hidden
  - lógos ≈ word
- The meaning can be paraphrased as „hidden word“
- This refers to the original intent of cryptology (i.e., data confidentiality protection)
- Today, the term is more broadly used
- It comprises other security-related purposes and applications (in addition to data confidentiality protection)

Kryptos is a sculpture by American artist Jim Sanborn (1990) located on the grounds of the Central Intelligence Agency (CIA) in Langley, Virginia

Introduction

Cryptology

- Examples
  - Data integrity protection
  - Enitity authentication
  - Data origin authentication
  - Access control
    (≈ conditional access, DRM, …)
  - Nonrepudiation
  - Accountability
  - Anonymity
  - Pseudonymity
- Some purposes and applications may be contradictory or even mutually exclusive (e.g., e-voting)

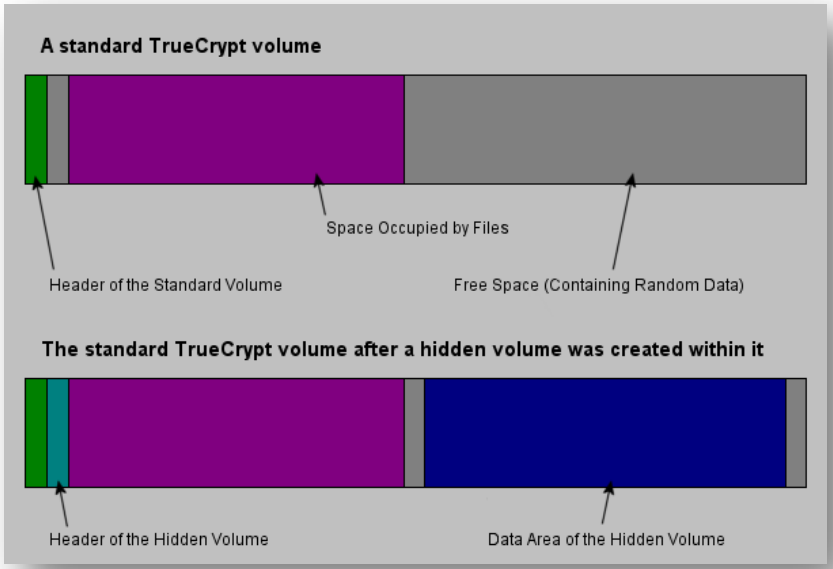
Introduction

Cryptology

- **Cryptology** is a mathematical science that comprises two branches of study (cf. RFC 2828)
  - **Cryptography** deals with transforming data to
    - render ist meaning unintelligible (i.e., hide its semantic content),
    - prevent its undetected alteration, or
    - prevent its unauthorized use
  - **Cryptanalysis** deals with the analysis of a cryptographic system in order to gain the knowledge needed to break or circumvent the pro-tection that the system is designed to provide
- The cryptanalyst is the (natural) antagonist of the cryptographer (there is an arms race going on)
- Cryptographic results are more openly discussed, but cryptanalyti-cal results are more frequently reported in the media

Introduction
Cryptology

- In some literature, cryptology also comprises **steganography**
- Real-world examples of steganographic techniques
  - Invisible ink
  - Tatoos under hair
  - ...
- Analogies (how to protect money)
  - Hide money (steganography)
  - Put money in a safe (cryptography)

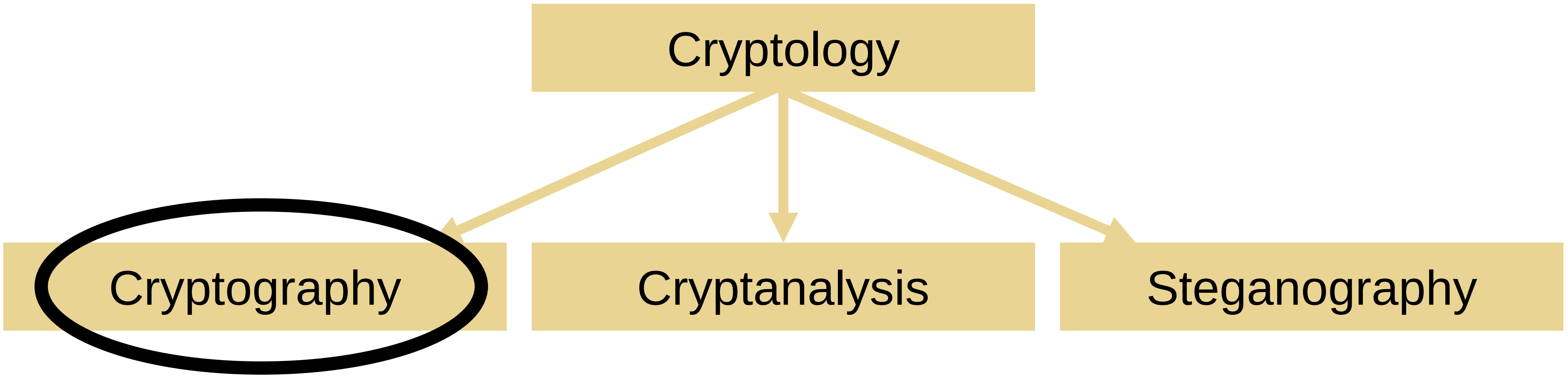

A standard TrueCrypt volume
Space Occupied by Files
Header of the Standard Volume
Free Space (Containing Random Data)
The standard TrueCrypt volume after a hidden volume was created within it
Header of the Hidden Volume
Data Area of the Hidden Volume

- Cryptographic and steganographic techniques are ideally com-bined (e.g., TrueCrypt hidden volumes)

Introduction

Cryptology

- In the digital world, there are many steganographic technqiues (i.e., techniques to hide the existence of information)
- Digital watermarking and fingerprinting employ similar techniques
  - **Watermarking** → Hide information about the data originator
  - **Fingerprinting** → Hide information about the data recipient



Introduction

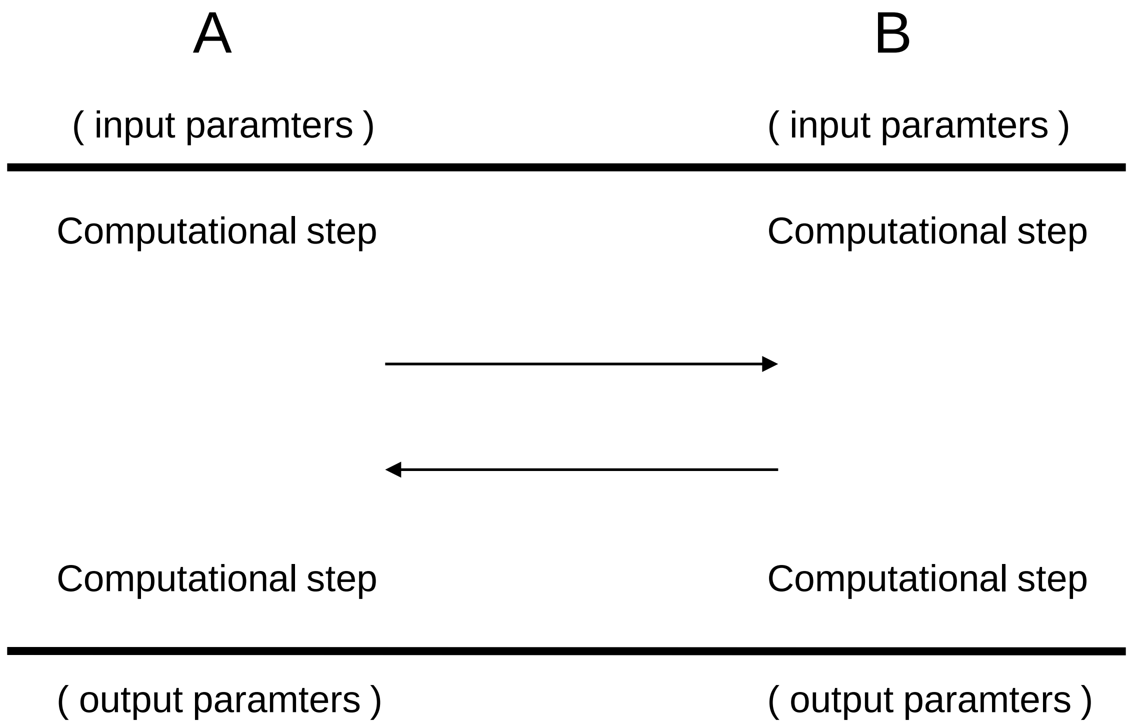
1.2 Cryptographic Systems

- According to RFC 2828, a **cryptographic system** (or **crypto-system**) is a set of algorithms together with the key management processes that support use of the algorithm in some application context
- In some literature, a cryptographic system is also called a **crypto-graphic scheme** (e.g., digital signature scheme)
- A distinction is made between an algorithm and a protocol
  - An **algorithm** is a well-defined computational procedure that takes a variable input and generates a corresponding output
  - A **protocol** is a distributed algorithm (i.e., an algorithm in which multiple entities take part)
- Hence, a protocol represents a distributed algorithm

Introduction

Cryptographic Systems

- Notation for protocols



Introduction

Cryptographic Systems

- Classes of cryptosystems
  - Unkeyed cryptosystem → no secret parameter
  - Secret key cryptosystem (symmetric) → secret parameters are shared among the participating entities
  - Public key cryptosystem (asymmetric) → secret parameters are not shared among the participating entities
- The goal of cryptography is to design, implement, and use crypto-systems that are reasonably „secure“
- To make precise statements about the security of a (crypto)system, one must formally define the term security
- Anything can be claimed secure, unless its meaning is defined and precisely nailed down

Introduction

Cryptographic Systems

- A **security definition** must answer (at least) two questions
  - What is the adversary one has in mind and against whom one wants to protect oneself?
    - How powerful is he or she?
    - What are his or her capabilities?
    - What attacks can he or she mount?
    - What is his or her a priori knowledge?
    - …
  - What is the task the adversary must solve in order to be successful (i.e., to break the system)?
- Strong security definitions are obtained if
  - The adversary is assumed to be as strong as possible
  - The task to be solved is assumed to be as simple as possible

Introduction

Cryptographic Systems

- Analogy: Security of a safe

– Adversary

– Task to solve

  - Open safe in 1 minute
  - Open safe in 5 minutes
  - Open safe in 15 minutes
  - Open safe in 1 hour
  - ...

Introduction

Cryptographic Systems

- Exercise 1-1: Security definitions

  1. Consider a soccer game and decide in which case your team is better

     a) Your team plays soccer against the world champion and he‘s not able to make a single goal

     b) Your team plays soccer against a group of schoolboys and they are not able to win the game

Introduction

Cryptographic Systems

- A cryptosystem is **secure** if a specified adversary cannot or is not able to solve a specified task
- Notions of security
  - If the adversary cannot solve the task, then one is in the realm of **unconditional** or **information-theoretic security** (→ probability and/or information theory)
  - If the adversary can in principle but is not able to solve the task, then one is in the realm of **conditional** or **computational security** (→ complexity theory)
- There are cryptosystems known to be secure in the strong sense (i.e., unconditionally secure), but there are no cryptosystems known to be secure in the weak sense (i.e., conditionally secure)
- Not even the existence of a conditionally secure cryptosystem has been proven so far (complexity theory deals with upper bounds)

Introduction

Cryptographic Systems

- In some literature, **provable security** is mentioned as (yet) another notion of security
- This goes back to the early days of public key cryptography, when Diffie and Hellman proposed the notion of a **complexity-based security proof** (aka **reduction proof**)
- The basic idea is to show that breaking a cryptosystem is **compu-tationally equivalent** to solving a hard (mathematical) problem
- Two directions
  - Hard problem can be solved $\Rightarrow$ Cryptosystem can be broken
  - Cryptosystem can be broken $\Rightarrow$ Hard problem can be solved
- Diffie and Hellman showed only the first direction for their key exchange protocol

Introduction

Cryptographic Systems

- The notion of provable security has fueled a lot of research since the mid-1970s
- Many (public key) cryptosystems have been proven secure in this sense
- But a complexity-based security proof (or reduction proof) is not absolute
- It is only relative to the assumed intractability of the underlying (mathematical) problem
- Hence, provable security is a special form of conditional or computational security
- The condition is the intractability of the underlying mathematical problem

Introduction

Cryptographic Systems

- In addition to the intractability assumption, people sometimes make the assumption that a cryptographic system (typically a cryptographic hash function) behaves like a **random function** (i.e., a function chosen at random from all possible functions)
- This sometimes allows a proof to be given in the **random oracle model** (as opposed to the **standard model**)
- The usefulness and suitability of the random oracle model is discussed controversially within the community

Introduction

Cryptographic Systems

- There are people who try to improve the security of (cryptographic) systems by keeping secret their design and internal working principles
- This approach (or state-of-mind) is known as **security through obscurity**
- Many systems that depend on security through obscurity do not provide adequate security
- Once their design and internal working principles get publicly known, they are broken rapidly

Introduction

Cryptographic Systems

- **Kerckhoffs‘ principle** states that a cryptographic system should be designed to be secure even when the adversary knows all the details of a system, except for the secret values (e.g., cryptogra-phic keys)
- The Kerckhoffs‘ principle is an appropriate and generally accepted principle for the design of cryptographic systems
- It is in line with the recommendation to assume a very powerful adversary
- But the principle is not without controversy in some application contexts

Introduction

Cryptographic Systems

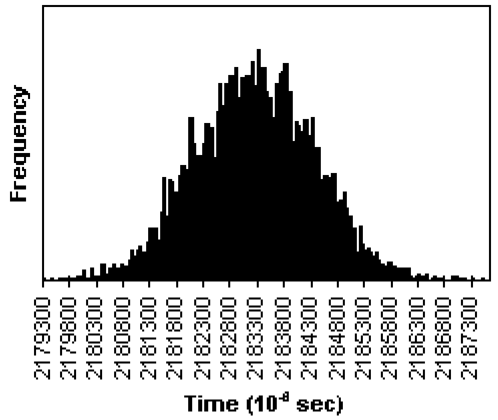
- An implementation of a (theoretically secure) cryptosystem need not be secure
- There are many attacks that can be mounted against a specific implementation of a (theoretically secure) cryptosystem
- Also, there are a number of **side channel attacks** that are surprisingly powerful
  - Timing attacks
  - Differential fault analysis (DFA)
  - Differential power analysis (DPA)
  - …
- More recently, research has started to address the notion of **leakage-resilient cryptography**

Introduction

Cryptographic Systems

FIGURE 2

RSAREF Modular Exponentiation Times



- Exercise 1-2: Protection against timing attacks
  1. What possibilities does one have to protect an implementation of a cryptosystem against timing attacks?
  2. What advantages and disadvantages do these possibilities have?

Introduction

1.3 Historical Background Information

- Until World War II, cryptography was considered to be an art and was primarily used in miltary and diplomacy
- Two developments and scientific achievements turned crypto-graphy from an art into a science
  - During World War II, **Claude E. Shannon** developed a mathematical theory of communication and a related theory of secrecy systems known as information theory

  - In the 1970s, **Whitfield Diffie** and **Martin Hellman** (as well as R**alph Merkle**) proposed the use of trapdoor (one-way) functions and came up with the notion of public key cryptography

Introduction

Historical Background Information

- Since the early 1990s, we have seen a wide deployment and commercialization of cryptography
- Today, many companies develop, market, and deploy cryptographic technologies and techniques, mechanisms, services, and/or products implemented in hardware and/or software
- There are many conferences (e.g., IACR conferences and workshops) and trade shows (e.g., RSA Security conferences) held on a regular basis
- Furthermore, the trade press is keen on publishing new cryptanalytical results

RSA CONFERENCE Where The World Talks Security

INTERNATIONAL ASSOCIATION FOR CRYPTOLOGIC RESEARCH

2 Cryptographic Systems (Overview)

2.1 Unkeyed Cryptosystems

2.2 Secret Key Cryptosystems

2.3 Public Key Cryptosystems

2.4 Final Remarks

Cryptographic Systems (Overview)

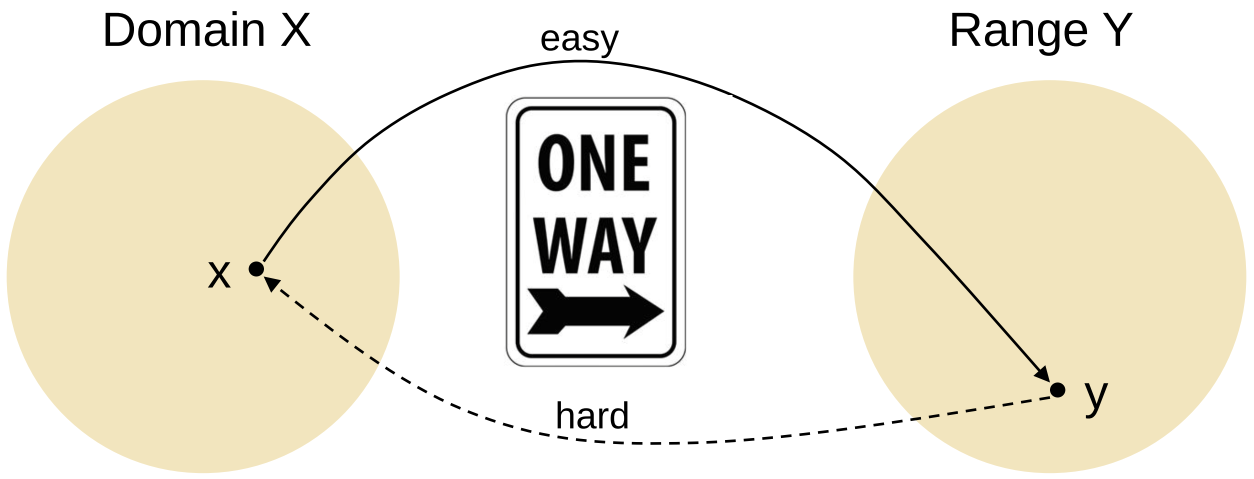
2.1 Unkeyed Cryptosystems

- One-Way Functions
- Cryptographic Hash Functions
- Random Bit Generators

Cryptographic Systems (Overview)

Unkeyed Cryptosystems – One-Way Functions

- The notion of a one-way function plays a pivotal role in contem-porary cryptography
- Informally speaking, a function f: X → Y is **one-way** if it is easy to compute but hard to invert



Cryptographic Systems (Overview)

Unkeyed Cryptosystems − One-Way Functions

- According to (computational) complexity theory
  - *Easy* means that the computation can be done efficiently
  - *Hard* means that it is not known how to do the computation efficiently (i.e., no efficient algorithm is known to exist)
- A computation is efficient if the expected running time of an algorithm that does the computation is bounded by a polynomial in the length of the input (i.e., $T(n) = (\ln n)^c$)
- A function $f: X \rightarrow Y$ is one-way if $f(x)$ can be computed efficiently for $x \in_R X$, but $f^{-1}(y)$ cannot be computed efficiently for $y \in_R Y$
- There are real-world examples of one-way functions
  - Loooking up a phone number in a telephone book
  - Smashing a bottle into pieces
  - Time-dependent processes (e.g., aging)

Cryptographic Systems (Overview)

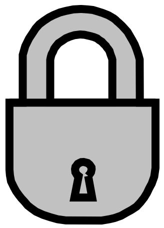
Unkeyed Cryptosystems − One-Way Functions

- In mathematics, there are only a few functions assumed (conjectured) to be one-way
  - Discrete exponentiation (Exp): $y = f(x) = g^x \pmod p$
  - Modular power (RSA): $y = f(x) = x^e \pmod n$
  - Modular square (Square): $y = f(x) = x^2 \pmod n$
- None of these functions has been proven to be one-way
- It is theoretically not even known whether one-way functions exist in the first place (i.e., a proof of existence is missing)

Cryptographic Systems (Overview)

Unkeyed Cryptosystems − One-Way Functions

- In public key cryptography, one often employs trapdoor (one-way) functions
- A one-way function $f: X \rightarrow Y$ is a **trapdoor (one-way) function** if there exists some extra information (trapdoor) with which f can be inverted efficiently, i.e., $f^{-1}(y)$ can be computed efficiently for $y \in_R Y$
- The mechanical analog of a trapdoor function is a padlock (the key represents the trapdoor)
  - Everybody can lock the padlock without the key
  - Nobody can open the padlock without the key

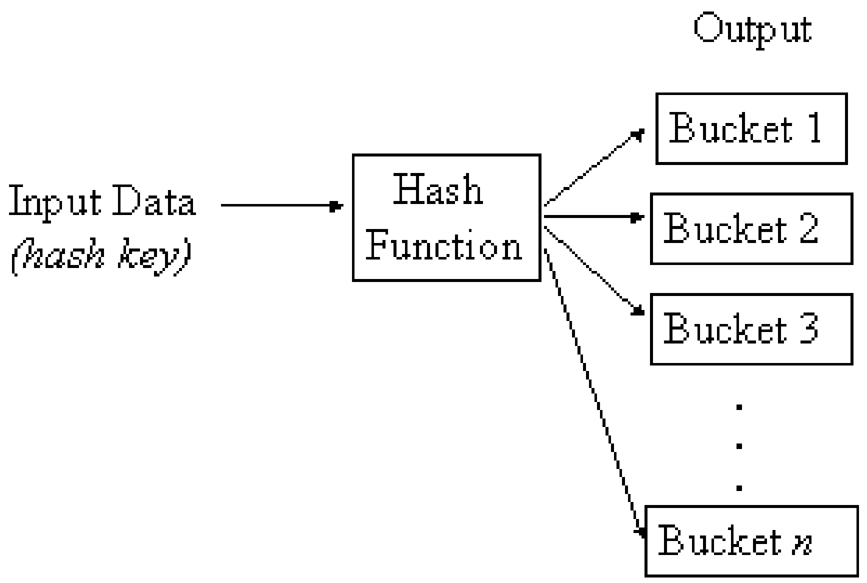


- One-way functions and trapdoor functions are the building blocks of public key cryptography
- To be mathematically precise (and apply complexity-theoretic arguments), one has to consider families of such functions

Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Cryptographic Hash Functions

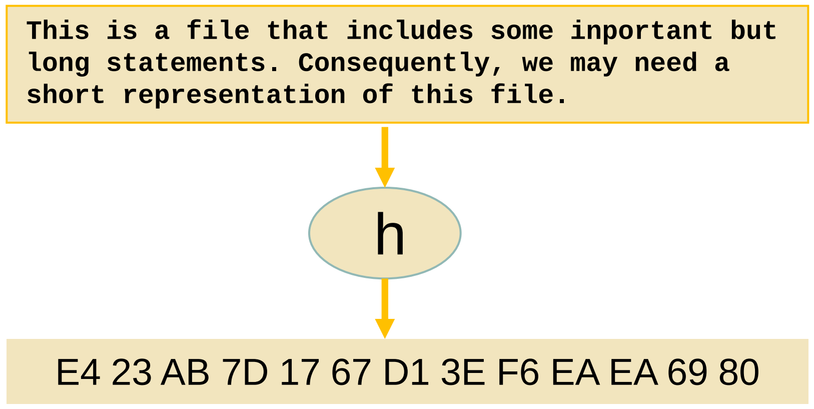
- Hash functions are frequently used and have many applications in computer science
- Informally speaking, a hash function is an efficiently computable function that takes an arbitrarily sized input (string) and generates an output (string) of fixed size
- Formally speaking, a hash function is an efficiently computable function $h: \Sigma^{n_{max}} \rightarrow \Sigma^{*n}$, where Σ is the input alphabet and Σ^* is the output alphabet
- Hence, h generates hash values of length n



Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Cryptographic Hash Functions

- In cryptography, one is interested in hash functions that are
  - Preimage resistent (≡ one way)
  - Second-preimage resistent
  - Collision resistent


This is a file that includes some inportant but long statements. Consequently, we may need a short representation of this file.
h
E4 23 AB 7D 17 67 D1 3E F6 EA EA 69 80

- A cryptographic hash function is a hash function that is either preimage resistent and second-preimage resistent or preimage resistent and collision resistent (preferred case)
- The output of a cryptographic hash function is a characteristic representation of the input string (message) representing a **fingerprint** or **digest**

Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Cryptographic Hash Functions

- The second-preimage and collision resistance properties ensure that it is computationally infeasible to find two (or more) messages that hash to the same value
- Exemplary cryptographic hash functions
  - MD5
  - SHA-1, SHA-2 family
  - SHA-3 (in progress)
- Finding collisions in cryptographic hash functions and finding functions that are inherently more collision resistent are active areas of research

Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Cryptographic Hash Functions

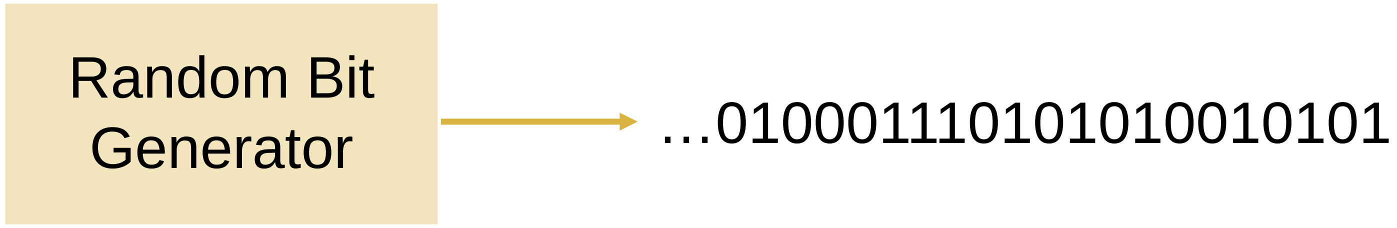
- Exercise 2-1: Cryptographic hash functions

1. Use CrypTool > Indiv. Procedures > Hash > Hash Demonstration… to visualize the effects of applying
   (a) MD5
   (b) SHA-1
   (c) RIPEMD-160
   to a message of your choice
2. Apply minor modifications to the message and try to predict the output behavior of the cryptographic hash function in use
3. Explain your observation(s)

Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Random Bit Generators

- **Randomness** is a fundamental ingredient for cryptography (and hence security)
- The generation of secret and unpredictable random quantities (i.e., random bits or random numbers) is at the core of most practically relevant cryptosystems
- A **random bit generator** is (an idealized model of) a device that has no input but outputs a sequence of statistically independent and unbiased bits



Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Random Bit Generators

- There are statistical tests that can be used to evaluate the (randomness) properties of a random bit generator (or the bit sequences that are generated, respectively)
- With regard to the design of a random bit generator, it is not undisputable whether randomness really exists
- Present knowledge in quantum physics suggests that randomness really exists
- But even if randomness exists, it is not immediately clear how to exploit it
  - There is no deterministic (i.e., computational) realization or implementation of a random bit generator
  - There are nondeterministic realizations and implementations of a random bit generator that employ some physical events or phenomena

Cryptographic Systems (Overview)

Unkeyed Cryptosystems − Random Bit Generators

- A (true) random bit generator requires a naturally occuring source of randomness
- Designing and implementing a device that exploits this source is an (engineering) challenge

DILBERT By Scott Adams
TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
www.dilbert.com
scottadams@aol.com
© 2001 United Feature Syndicate, Inc.

Cryptographic Systems (Overview)

2.2 Secret Key Cryptosystems

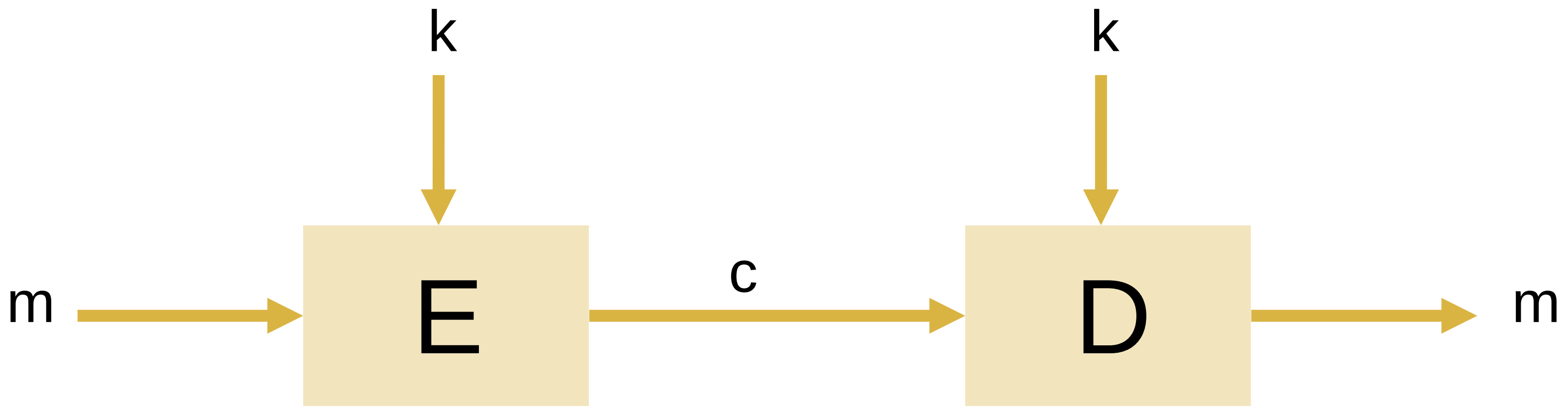
- Symmetric Encryption Systems
- Message Authentication Codes
- Pseudorandom Bit Generators (PRBGs)
- Pseudorandom Functions (PRFs)

(c) http://www.gpg4win.org/doc/images-compendium/secret-key-exchange.png

Cryptographic Systems (Overview)

Secret Key Cryptosystems – Symmetric Encryption ...

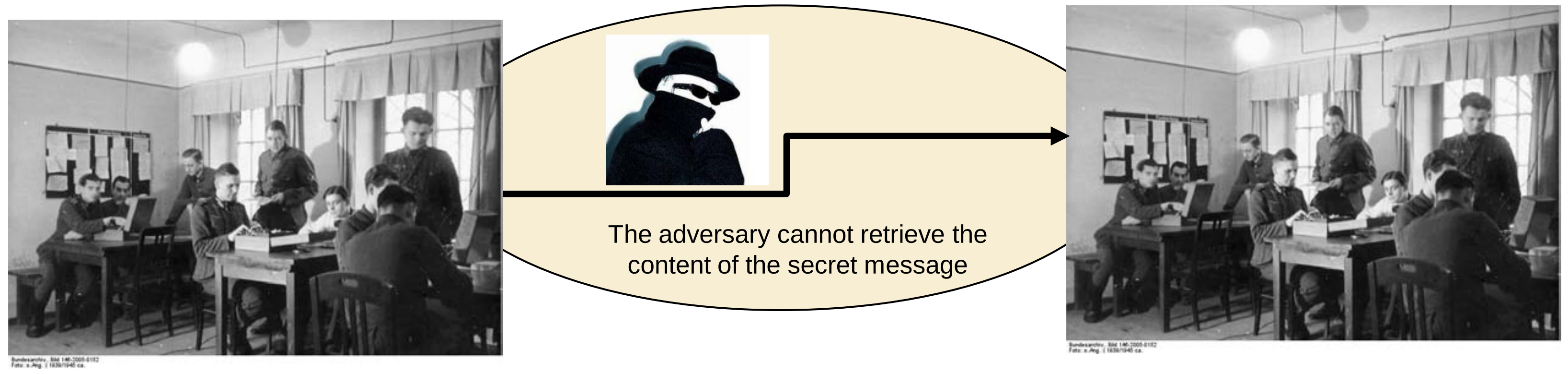
- A **symmetric encryption system** can be used to encrypt and decrypt data with a secret key k
  - **Encryption** is the process that turns a plaintext message m into a ciphertext c
  - **Decryption** is the reverse process that turns a ciphertext c into a plaintext message m



Cryptographic Systems (Overview)

Secret Key Cryptosystems – Symmetric Encryption ...

- The use case is to transmit a secret message from A to B


The adversary cannot retrieve the content of the secret message

- Examples
  - DES, DESX, 3DES, ...
  - RC2, RC4, RC5, RC6, ...
  - IDEA
  - AES
  - ...

Cryptographic Systems (Overview)

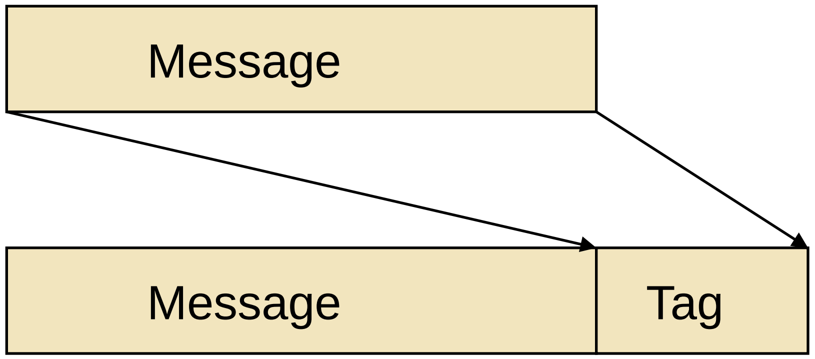
Secret Key Cryptosystems – Symmetric Encryption ...

- To reasonably argue about the security of a symmetric encryption system, one must specify the adversary and the task he or she must solve
  - Typically, the adversary is polynomially bounded and can mount specific attacks
    - Ciphertext-only attack
    - Known-plaintext attack
    - (Adaptive) chosen-plaintext attack (CPA)
    - (Adaptive) chosen-ciphertext attack (CCA / CCA2)
  - The task is to decrypt a ciphertext or to determine plaintexts that are more probable than others

Cryptographic Systems (Overview)

Secret Key Cryptosystems – Message Authentication ...

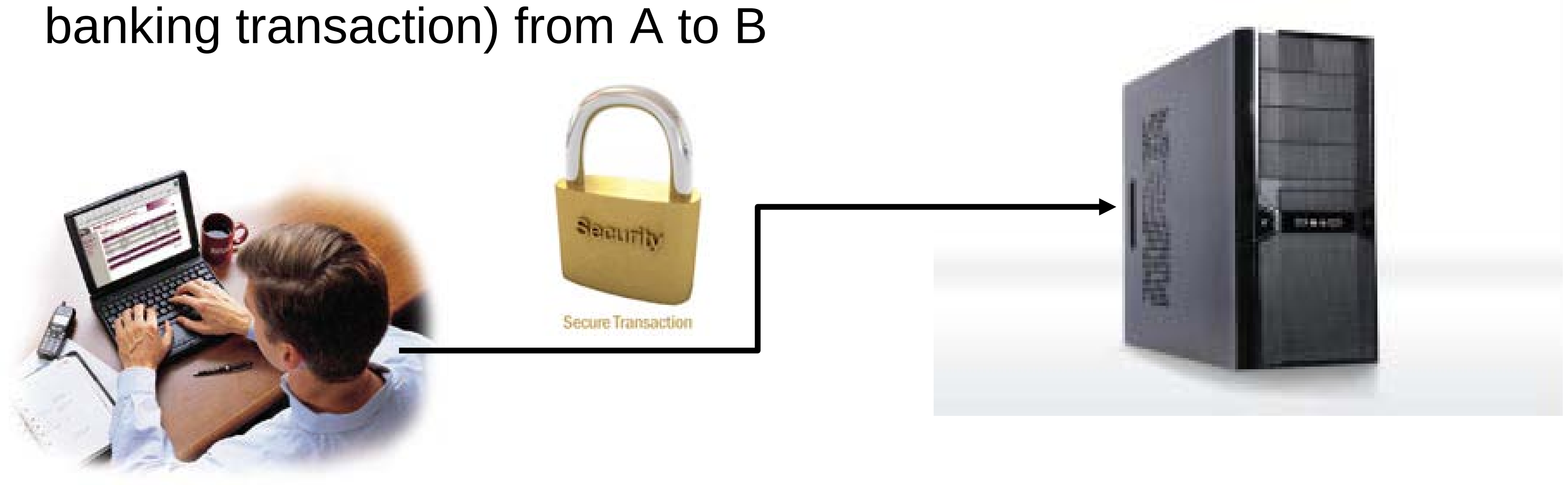
- It is sometimes argued that encrypting messages also protects their authenticity and integrity (in addition to their confidentiality)
- This argument is flawed and confidentiality protection and authenticity and integrity protection are different pairs of shoes
- To protect the authenticity and integrity of a message, the sender can append an **authentication tag** to the message and the recipient can verify it



Cryptographic Systems (Overview)

Secret Key Cryptosystems – Message Authentication ...

- The use case is to securely transmit a message (e.g., an online banking transaction) from A to B



- Possibilities to compute and verify authentication tags
  - Public key cryptography (→ digital signatures)
  - Secret key cryptography (→ message authentication codes)

Cryptographic Systems (Overview)

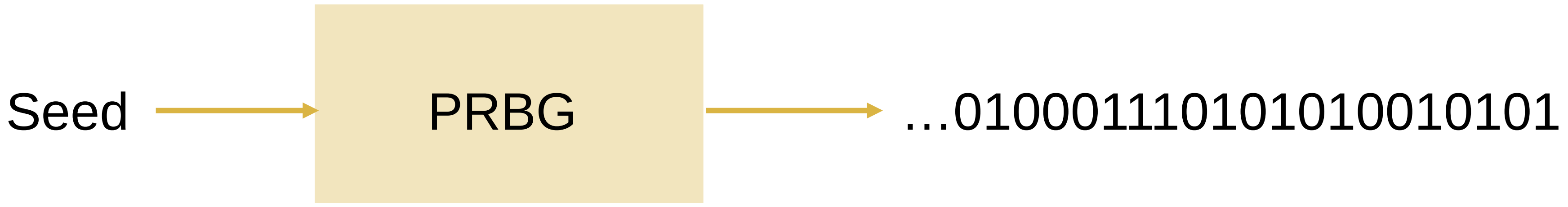
Secret Key Cryptosystems – Message Authentication ...

- A **message authentication code (MAC)** is an authentication tag computed and verified with a secret parameter (i.e., secret key)
- A **message authentication system** is a system that can be used to compute and verify MACs
- To reasonably argue about the security of a message authentication system, one must specify the adversary and the task he or she must solve
- Informally speaking, one wants to avoid an adversary being able to illegitimately generate a valid-looking MAC
- Many (unconditionally and conditionally secure) message authentication systems have been proposed
- In practice, the **HMAC construction** is most frequently used

Cryptographic Systems (Overview)

Secret Key Cryptosystems – PRBGs

- Remember that a random bit generator cannot be implemented deterministically
- A **PRBG** is an efficient deterministic algorithm that takes as input a random binary sequence of length k (seed) and generates as output another binary sequence (pseudorandom bit sequence) of possibly infinite length l >> k
- The use case is to generate randomly-looking but not predictable keying material from a relatively short seed



Cryptographic Systems (Overview)

Secret Key Cryptosystems – PRBGs

- Unlike a random bit generator, a **PRBG** represents a deterministic algorithm and has an input (namely the seed)
- Hence, it must be implemented as a **finite state machine (FSM)** and it generates a bit sequence that is cyclic (potentially large cycle)
- One cannot require that the bits in a pseudorandomly generated bit sequence are truly random – only that they appear to be so (i.e., they cannot be told apart from truly randomly generated bits)
- Similar to the case of a random bit generator, statistical tests can be used to evaluate the randomness properties of the bit sequen-ces generated by a PRBG
- The bottom line is that a PRBG outputs bit sequences that have specific (randomness) properties

Cryptographic Systems (Overview)

Secret Key Cryptosystems – PRFs

- A **random function** (or **random oracle**) is a function $f: X \rightarrow Y$ that is randomly chosen from the set of all possible mappings from domain X to range Y
- For $x \in X$, a random function outputs an arbitrarily chosen (but always the same) $y = f(x) \in f(X) \subseteq Y$
- A **pseudorandom function (PRF) family** is a collection of efficiently-computable functions that can be used to emulate a random function, i.e., no efficient algorithm can distinguish between a function chosen randomly from the PRF family and a true random function

Cryptographic Systems (Overview)

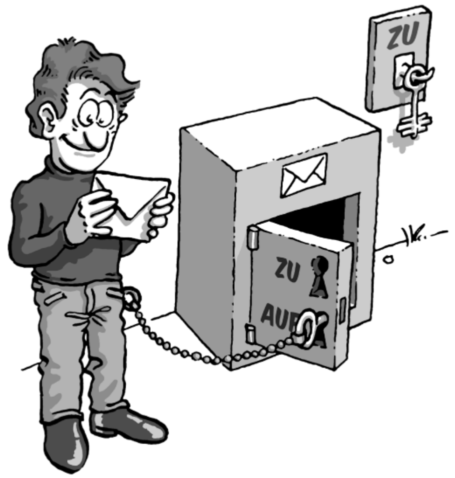
Secret Key Cryptosystems – PRFs

- In spite of their different properties, PRBGs and PRF families are related to each other
  - A PRF family can be used to construct a PRBG
  - A PRBG can be used to construct a PRF family
- Note that PRF families are mainly used in theory (in formal secuirty proofs), and that they are not intended to be implemen-ted in practice
- This is in contrast to PRBGs that are frequently used in practice

Cryptographic Systems (Overview)

2.3 Public Key Cryptosystems

- Asymmetric Encryption Systems
- Digital Signature Systems (DSSs)
- Key Agreement

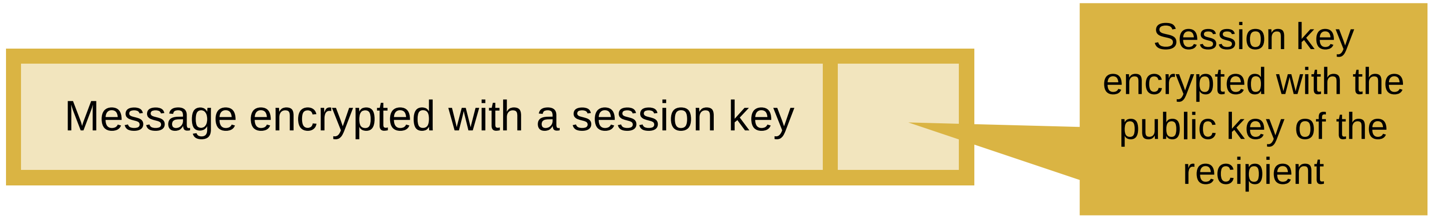


(c) http://www.gpg4win.org/doc/images-compendium/secret-key-exchange.png

Cryptographic Systems (Overview)

Public Key Cryptosystems

- In a public key cryptosystem, each entity holds a pair of mathematically related keys (**public key** k and **private key** k^{-1})
- A necessary (but usually not sufficient) prerequisite for a public key cryptosystem to be secure is that it is computationally infeasible to compute k^{-1} from k
- As such, k can be published (e.g., in a directory)
- Public key cryptography is computationally much less efficient than secret key cryptography
- In practice, one combines public and secret key cryptography in **hybrid systems** (e.g., secure messaging)


Message encrypted with a session key
Session key encrypted with the public key of the recipient

Cryptographic Systems (Overview)

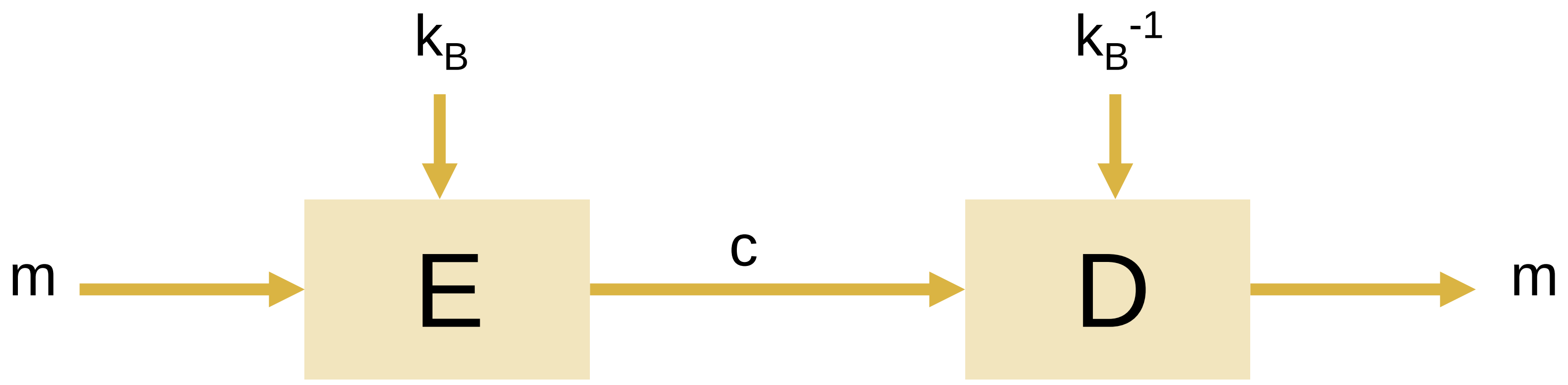
Public Key Cryptosystems

- In public key cryptography, it is common to specifiy a system using a set of algorithms
- One algorithm typically refers to the generation of the key pairs
- The other algorithms depend on the system in use

Cryptographic Systems (Overview)

Public Key Cryptosystems – Asymmetric Encryption ...

- Similar to a symmetric encryption system, an asymmetric encryption system can be used to encrypt and decrypt data (messages)
- It requires a (family of) trapdoor (one-way) functions
  - The public key k_B represents the one-way function
  - The private key k_B^{-1} represents the trapdoor (to the one-way function)



Cryptographic Systems (Overview)

Public Key Cryptosystems – Asymmetric Encryption ...

- The use case is slightly different than in the symmetric case, because an asymmetric encryption system allows the sender to secretly transmit a message the recipient without prior key agreement
- The sender just takes the public key of the recipient and subjects the message to the respective one-way function
- Noboday can retrieve the message without inverting the one-way function (and this is assumed to be computationally intractable)

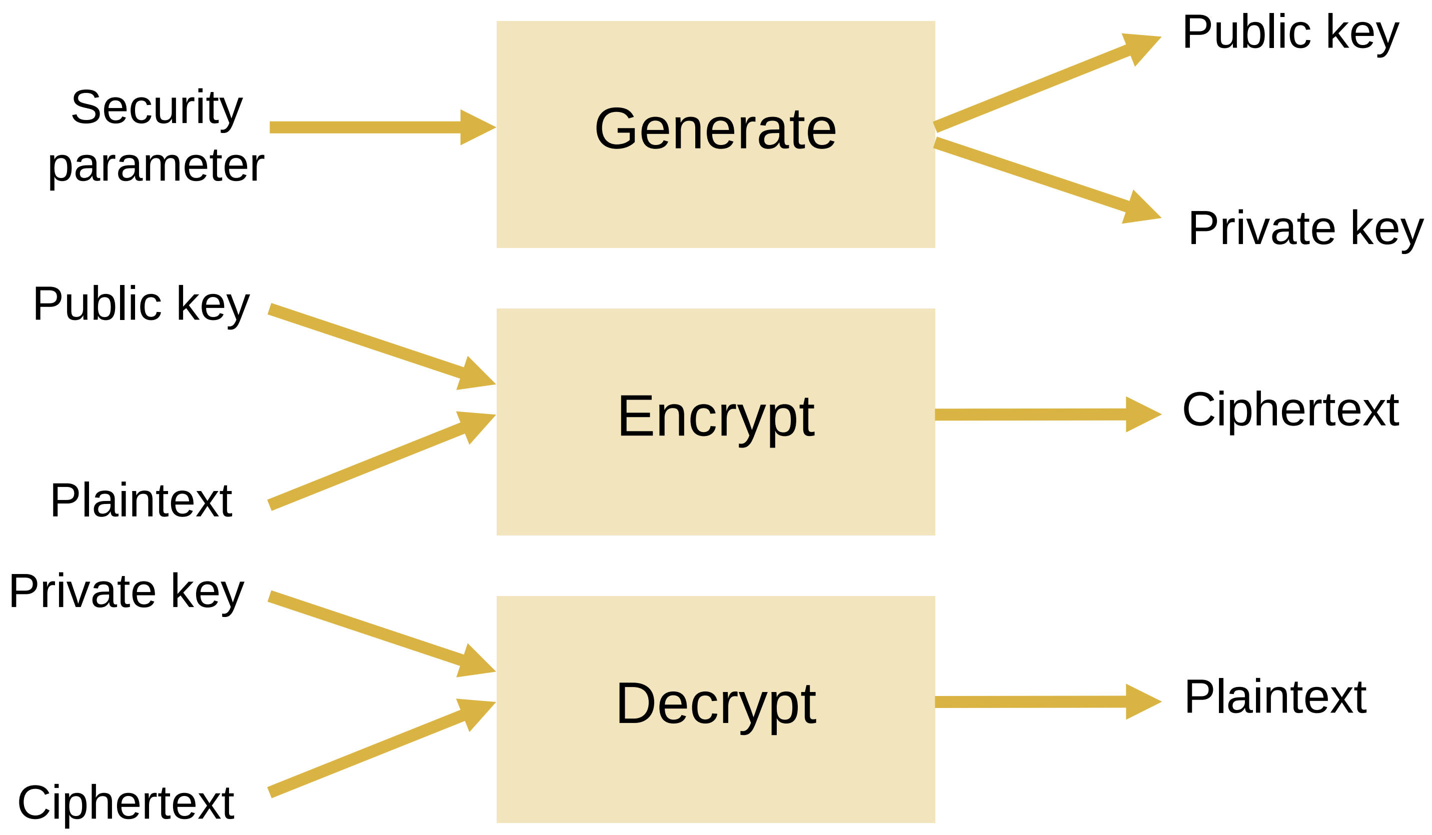
Cryptographic Systems (Overview)

Public Key Cryptosystems – Asymmetric Encryption ...

- An **asymmetric encryption system** consists of the following 3 efficiently computable functions or algorithms
  - **Generate(1^l)** is a probabilistic key generation algorithm that generates a public key pair (k,k^{-1}) based on the security parameter l ($\cong$ key length)
  - **Encrypt(k,m)** is a deterministic or probabilistic encryption algorithm that generates a ciphertext c, i.e., c = Encrypt(k,m)
  - **Decrypt(k^{-1},c)** is a deterministic decryption algorithm that is inverse to Encrypt(k,m) and generates the plaintext message, i.e., m = Decrypt(k^{-1},c)

Cryptographic Systems (Overview)

Public Key Cryptosystems – Asymmetric Encryption ...



Cryptographic Systems (Overview)

Public Key Cryptosystems – Asymmetric Encryption ...

- Discussing the security of asymmetric encryption is more involved than in the symmetric case
- This is because the encryption key is public and can be used to mount chosen plaintext attacks (CPAs) at will
- To defeat CPAs and various types of chosen ciphertext attacks (CCAs), different notions of security have been explored in the recent past
  - Semantic security
  - Indistinguishability of ciphertexts
  - Nonmalleability
  - …

Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs

- Technically speaking, a digital signature is
  - A value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data‘s origin and integrity (RFC 2828)
  - Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery, e.g. by the recipient (ISO/IEC 7498-2)
- Digital signatures can be used to protect the authenticity and integrity of messages, and to provide nonrepudiation services
- With the proliferation of e-commerce, digital signatures and the legislation thereof have become important and timely topics
- Most countries have put in place a digital signature law

Cryptographic Systems (Overview)

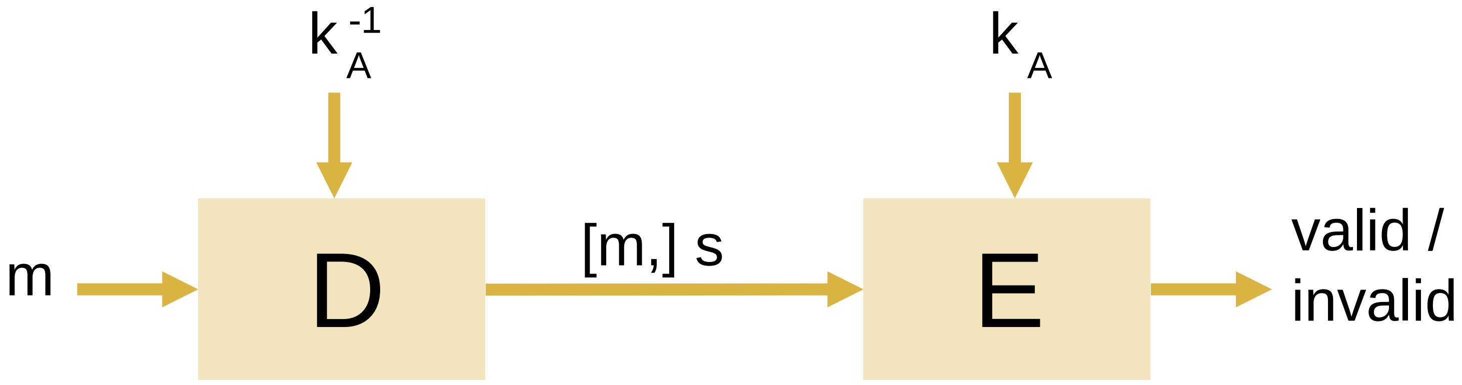
Public Key Cryptosystems – DSSs

- The use case of a DSS is similar to the one of a message authentication system
- But digital signatures are able to provide non-repudiation
- Types of digital signatures
  - **Digital signature with appendix** → Digital signature is appended to the data unit (ISO/IEC 14888)
  - **Digital signature giving message recovery** → Data unit is crypto-graphically transformed in a way that it represents both the data unit and the digital signature (ISO/IEC 9796)

Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs

- Working principles
  - The signatory A uses its private key k_A^{-1} to compute the signature $s = D_A(m)$
  - The verifier uses A's public key k_A to verify the signature s (or to recover the message m, respectively) → public verifiability



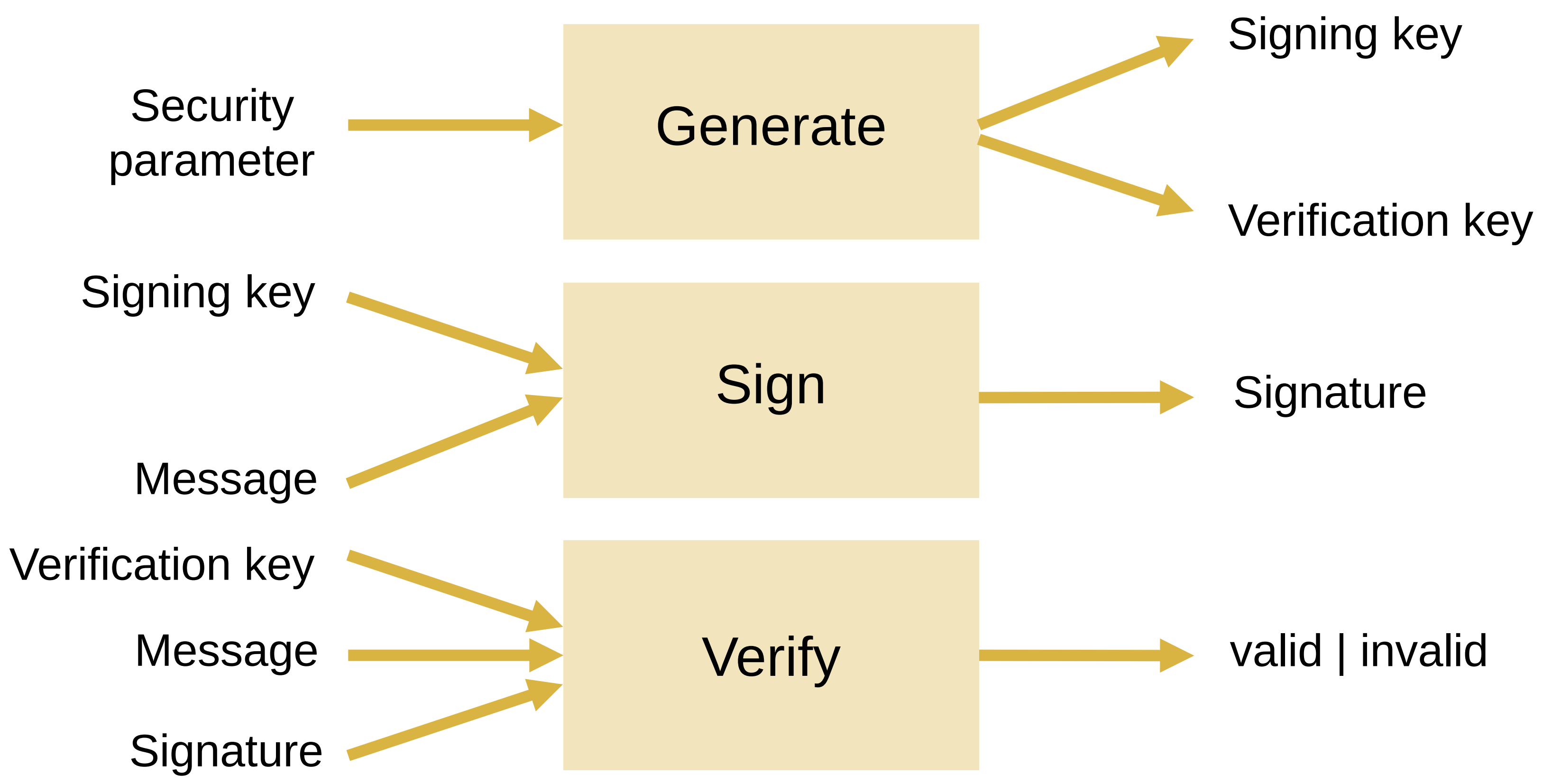
Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs

- A **DSS with appendix** consists of 3 efficiently computable functions or algorithms
  - **Generate(1^l)** is a probabilistic key generation algorithm
  - **Sign(k^{-1},m)** is a deterministic or probabilistic signature generation algorithm that outputs a signature s = Sign(k^{-1},m)
  - **Verify(k,m,s)** is a deterministic signature verification algorithm that yields valid iff s is a valid signature for m and k

Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs



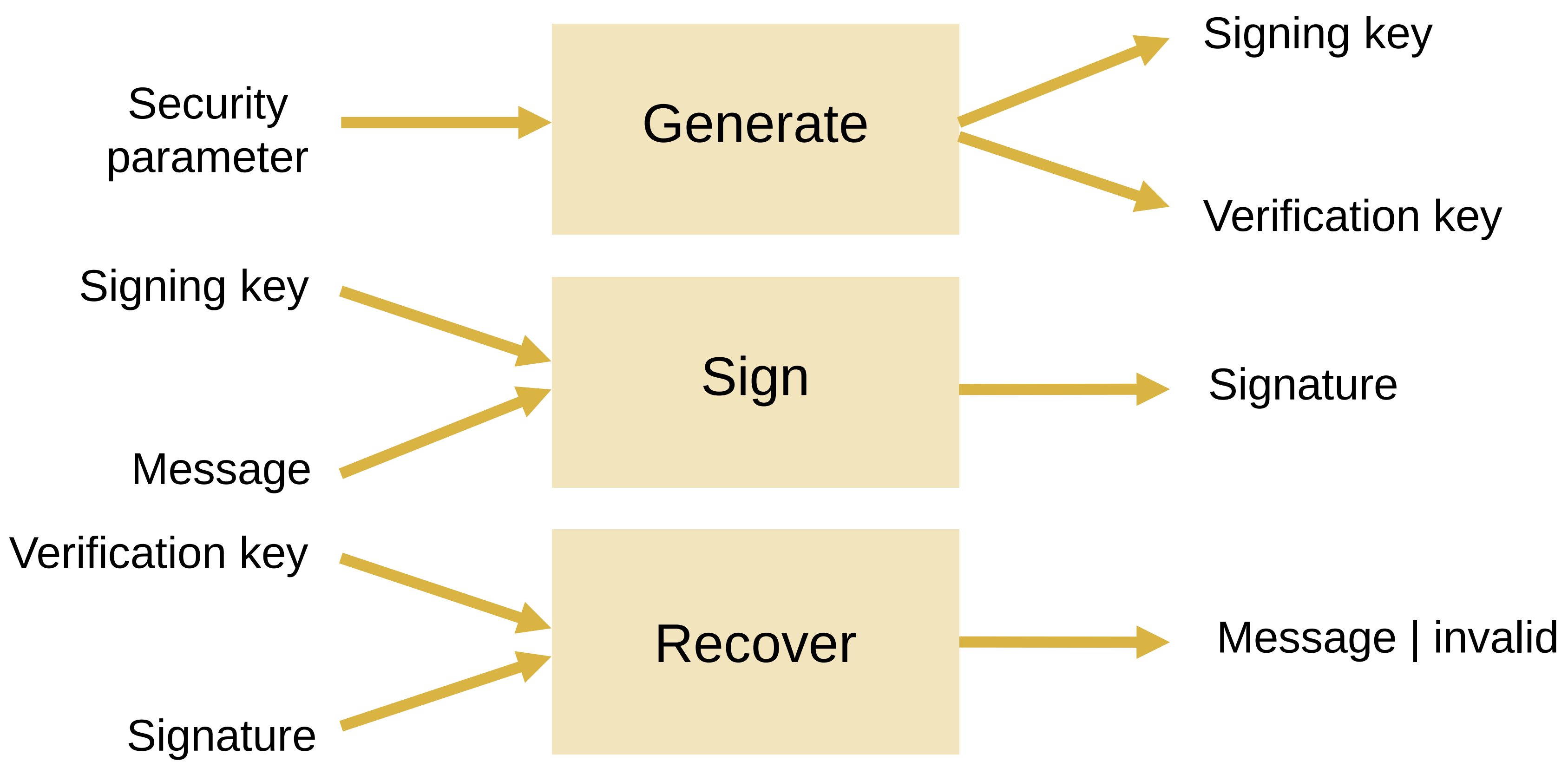
Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs

- A DSS giving message recovery consists of 3 efficiently computable functions or algorithms
  - **Generate(1^l)** is a probabilistic key generation algorithm
  - **Sign(k^{-1},m)** is a deterministic or probabilistic signature generation algorithm that outputs a signature s giving message recovery
  - **Recover(k,s)** is a deterministic message recovery algorithm that outputs either the message or a notification indicating that s is an invalid signature

Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs



Cryptographic Systems (Overview)

Public Key Cryptosystems – DSSs

- Discussing the security of a DSS is involved
- It must be computationally infeasible to forge a signature, i.e., illegitimatly generate a valid-looking signature
- There are, however, different attacks to consider and different ways to forge a signture (selective vs. existential forgeries)
- The bottom line is that there are different notions of security for DSSs
- Some of these notions are very strong, but they only exist in theory

Cryptographic Systems (Overview)

Public Key Cryptosystems – Key Agreement

- If two (or more) entities want to make use of secret key cryptography, then they must share a secret (key)
- In large systems, many keys must be generated, stored, managed, and destroyed in a secure way
- For n entities, there are

$$\left[\begin{matrix} n \\ 2 \end{matrix}\right] = \frac{n(n-1)}{1 \cdot 2} = \frac{n^2-n}{1 \cdot 2}$$

  keys (e.g., 499,500 keys for n = 1,000)
- The resulting (scalability) problem is known as the **n^2-problem**

Cryptographic Systems (Overview)

Public Key Cryptosystems – Key Agreement

- Approaches to solve the n^2-problem
  - Key distribution center (e.g., Kerberos)
  - Key establishment protocol
- Types of key establishment protocols
  - Key distribution protocols (e.g., SSL/TLS handshake with RSA-based session key establishment)
  - Key agreement protocols (e.g., Diffie-Hellman key exchange)
- As mentioned before, the Diffie-Hellman key exchange protocol gave birth to public key cryptography

Cryptographic Systems (Overview)

2.4 Final Remarks

- The cryptographic community is in a phase of consolidation
- A major theme is to better understand and formally define the notion of security, and to prove that particular cryptosystems aresecure in this sense
- In the remaining parts of this seminar, the cryptographic systems that we have overviewed are refined, more precisely defined, discussed, and put into perspective
- The aim is to provide a comprehensive overview and a better understanding of contemporary cryptography and the crypto-systems in current use

Module 2

Secret Key Cryptography

3 Symmetric Encryption Systems

Symmetric Encryption Systems

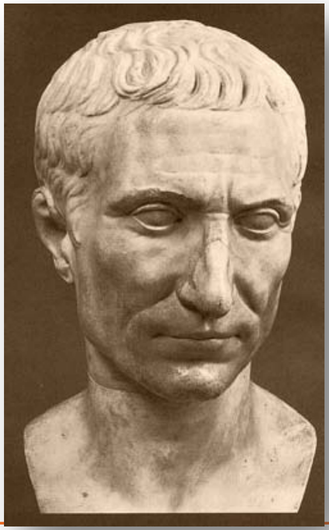
3.1 Introduction

- Mathematically speaking, a **symmetric encryption system** (or **cipher**) consists of 5 components
  - Plaintext message space M
  - Ciphertext space C
  - Key space K
  - Family E = $\{E_k: k \in K\}$ of (probabilistic) encryption functions $E_k: M \rightarrow C$
  - Family D = $\{D_k: k \in K\}$ of decryption functions $D_k: C \rightarrow M$
- D_k and E_k must be inverse to each other, i.e., $D_k(E_k(m)) = E_k(D_k(m)) = m$ ($\forall k \in K, m \in M$)
- In a typical setting
  - $M = C = \Sigma^*$
  - $K = \Sigma^l$ for some fixed key length l (e.g., $l = 128$)

Symmetric Encryption Systems

Introduction

- For $\Sigma = \{A,\ldots,Z\}$, Σ^* consists of all words that can be constructed with the capital letters A to Z
- These letters can be associated with $\mathbf{Z}_{26} = \{0,\ldots,25\}$
- There is a bijective map from $\{A,\ldots,Z\}$ to $\mathbf{Z}_{26} = \{0,\ldots,25\}$, i.e., $\{A,\ldots,Z\} \cong \mathbf{Z}_{26}$, and hence one can work with $\{A,\ldots,Z\}$ or $\mathbf{Z}_{26}$
- For $\Sigma = \{A,\ldots,Z\} \cong \mathbf{Z}_{26}$ and $M = C = K = \Sigma^*$, the **additive cipher** can be defined as follows
  - Encryption function E_k: $M \rightarrow C$; $m \rightarrow m + k \pmod{26} = c$
  - Decryption function D_k: $C \rightarrow M$; $c \rightarrow c - k \pmod{26} = m$
- For k = 3, the additive cipher is known as the **Caesar cipher**



Symmetric Encryption Systems

Introduction

- Exercise 3-1: Additive cipher

  1. Work through the CrypTool animation of the Caesar cipher (CrypTool > Indiv. Procedures > Visualization of Algorithms > Caesar…)
  2. Use CrypTool to encrypt a text file with an additive cipher using different keys (CrypTool > Crypt/Decrypt > Symmetric (classic) > Caesar / Rot-13…)
  3. Discuss the advantages and disadvabtages, as well as the possibilities to break the cipher
  4. Use CrypTool to actually break the cipher (CrypTool > Analysis > Symmetric Encryption (classic) > Ciphertext only > Caesar)

Symmetric Encryption Systems

Introduction

- Similar to the additive cipher, one can define a multiplicative cipher or combine an additive and a multiplicative cipher in an affine cipher
- In an **affine cipher**, *K* consists of all pairs $(a,b) \in \mathbf{Z}_{26}^2$ with gcd(a,26) = 1 → *K* has $\Phi(26) \cdot 26 = 12 \cdot 26 = 312$ elements
  - Encryption function $E_k: M \rightarrow C; m \rightarrow am + b \pmod{26} = c$
  - Decryption function $D_k: C \rightarrow M; c \rightarrow a^{-1}(c - b) \pmod{26} = m$
- Toy example
  - $a = 3$, $b = 1$, $a^{-1} \bmod 26 = 9$ $(3 \cdot 9 = 27 \equiv 1 \pmod{26})$
  - Encryption of $m = 2$: $c = am + b \pmod{26} = 3 \cdot 2 + 1 \pmod{26} = 7$
  - Decryption of $c = 7$: $m = a^{-1}(c - b) \pmod{26}$
    $= 9(7-1) = 9 \cdot 6 = 54 \pmod{26} = 2$

Symmetric Encryption Systems

Introduction

- An affine cipher can be broken with a known plaintext attack and only 2 plaintext-ciphertext pairs (m,c)
  - If an adversary knows (m1,c1) = (f,q) = (5,16) and (m2,c2) = (t,g) = (19,6), then he can set up and solve a system of two equivalences

$$a5 + b \equiv 16 \pmod{26}$$
$$a19 + b \equiv 6 \pmod{26}$$

  - The first equivalence can be rewritten as $b \equiv 16 - 5a \pmod{26}$ and used in the second equivalence

$$19a + b \equiv 6 \pmod{26}$$
$$19a + 16 - 5a \equiv 6 \pmod{26}$$
$$14a + 16 \equiv 6 \pmod{26}$$

Symmetric Encryption Systems

Introduction

– Consequently,

$$14a \equiv -10 \equiv 16 \pmod{26}$$

$$7a \equiv 8 \pmod{13}$$

– By multiplying either side with the multiplicative inverse element of 7 modulo 13 (which is 2), one gets

$$a \equiv 2 \cdot 8 = 16 \equiv 3 \pmod{13}$$

– Hence, $a = 3$ and $b = 1$ (since $b \equiv 16 - 5a \pmod{26}$)

– Similar to the legitimate recipient, the adversary can now compute $D_{(a,b)}$

Symmetric Encryption Systems

Introduction

- $\Sigma = \{A,\ldots,Z\} \cong \mathbf{Z}_{26}$ is a good choice for human beings
- If computer systems are used for encryption and decryption, then it is better and more appropriate to use $\Sigma = \mathbf{Z}_2 = \{0,1\} \cong GF(2)$
- Additive, multiplicative, and affine ciphers represent mono-alphabetic substitution ciphers (i.e., one single substitution alphabet is used)
- Such ciphers can be broken with frequency analyses
- Alternatives
  - Homophonic substitution ciphers
  - Polyalphabetic substitution ciphers (e.g., Vignère cipher)

Symmetric Encryption Systems

Introduction

- Exercise 3-2: Homophonic substitution cipher
  1. Use CrypTool to encrypt an arbitrary text file with a homophonic substitution cipher (CrypTool > Crypt/Decrypt > Symmetric (classic) > Homophone…)
  2. Discuss the advantages and disadvantages of homophonic substitution ciphers

Symmetric Encryption Systems

Introduction

- Exercise 3-3: Polyalphabetic substitution cipher
    1. Work through the CrypTool animation of the Vigenère cipher (CrypTool > Indiv. Procedures > Visualization of Algorithms > Vigenère)
    2. Use CrypTool to encrypt an arbitrary text file with the Vigenère cipher (CrypTool > Crypt/Decrypt > Symmetric (classic) > Vigenère…)
    3. Discuss the advantages and disadvantages of the Vigenère cipher
    4. Use CrypTool to break the Vigenère cipher (CrypTool > Analysis > Symmetric Encryption (classic) > Ciphertext only > Vigenère…)

Symmetric Encryption Systems

Introduction

- Homophonic and polyalphabetic substitution ciphers can be broken with more sophisticated cryptanalytical methods
- For example, in the case of the Vigenère cipher, the number k of substitution alphabets (representing the key length) must be deter-mined first
- Knwoing k, a given ciphertext can be divided up into k ciphertexts, each of them encrypted with a mono-alphabetic substitution cipher
- Statistical tests to determine k
  - Kasiski test (Friedrich Wilhelm Kasiski, 1863)
  - Friedman test aka index of coincidence (William F. Friedman, 1925)

Symmetric Encryption Systems

Introduction

- Similar techniques apply to iterated polyalphabetic substitution ciphers, such as rotor machines (e.g., Enig-ma used in World War II)
- An Enigma Java applet is available at http://homepages.tesco.net/~andycarlson/enigma/enigma_j.html

Symmetric Encryption Systems

Introduction

- Every practically relevant symmetric encryption system processes plaintext messages unit by unit
- A unit may be a bit, a block of bits (e.g., a byte), or a block of bytes
- The symmetric encryption system may be implemented as a **finite state machine (FSM)**, i.e., the ith ciphertext unit depends on
  - ith plaintext unit
  - Secret key
  - Possibly some internal state (stream ciphers)

Symmetric Encryption Systems

Introduction

- Depending on the existence and use of internal state, block and stream ciphers are distinguished
  - In a **block cipher**, the encrypting and decrypting devices have no internal state, i.e., the i^{th} ciphertext unit only depends on the i^{th} plaintext unit and the secret key
  - In a **stream cipher**, the encrypting and decrypting devices have internal state, i.e., the i^{th} ciphertext unit depends on the i^{th} plaintext unit, the secret key, and some internal state

Symmetric Encryption Systems

Introduction

- Classes of stream ciphers
  - In a synchronous or additive stream cipher, the state does not depend on previously generated ciphertext units
  - In a nonsynchronous or self-synchronizing stream cipher, the state also depends on some (or all) previously generated ciphertext units
- The distinction between block ciphers and stream ciphers is less precise than one might expect
- There are modes of operation that turn a block cipher into a stream cipher

Symmetric Encryption Systems

Introduction

- The most straightforward attack against a symmetric encryption system is a **brute-force attack** or **exhaustive key search**, i.e., an adversary tries to decrypt a ciphertext with every possible key $k \in K$
- On the average, half of the key space (i.e., $|K|/2$)must be searched through
- A brute-force attack requires that the adversary is able to decide whether he has found the correct plaintext message (or key)
  - Adversary knows plaintext (i.e., known-plaintext attack)
  - Plaintext message is written in a specific language
  - More generally, plaintext messages contain enough redundancy to tell them apart from gibberish
- The plaintext message may be compressed or specifically en-coded (e.g., Base-64) and only look like gibberish

Symmetric Encryption Systems

Introduction

- Shannon‘s evaluation criteria for symmetric encryption systems
  - Amount of secrecy
  - Size of key
  - Complexity of enciphering and deciphering operations
  - Propagation of errors
  - Expansion of messages

Symmetric Encryption Systems

3.2 Block Ciphers

- A block cipher maps plaintext message blocks of length n to ciphertext blocks of typically the same length (i.e., $M = C = \Sigma^n$)
- A block cipher then represents a family of permutations $\pi \in K$
  - Encryption function $E_\pi: \Sigma^n \rightarrow \Sigma^n$ $(m \rightarrow \pi(m))$
  - Decryption function $D_\pi: \Sigma^n \rightarrow \Sigma^n$ $(c \rightarrow \pi^{-1}(c))$
- The key space K comprises all permutations over Σ^n, i.e., $K = P(\Sigma^n)$)
- There are $|P(\Sigma^n)| = (|\Sigma|^n)!$ possible permutations that can be used as block ciphers (with block length n)
- For example, for $\Sigma = \{0,1\}$ and block length n, there are (2n)! possible permutations
- This functions grows tremendously

Symmetric Encryption Systems

Block Ciphers

- For a typical block length of 64 bits, there are

  $$2^{64}! = 18{,}446{,}744,\ 073{,}709,\ 551{,}616!$$

  possible permutations (this number requires $>2^{69}$ bits)
- Hence, if one wants to specify a particular permutation, then one has to number the permutations and use an index of $>2^{69}$ bits to serve as key
- This is impractical
- Block ciphers are designed to take a reasonably long key (e.g., rather 69 than 2^{69} bits) and generate a mapping that looks ran-dom to someone who does not know the key
- To study the security of a block cipher, one must analyze the algebraic properties of this mapping

Symmetric Encryption Systems

Block Ciphers

- In general, a (symmetric) encryption system is to provide a maximum level of confusion and diffusion
  - **Confusion** is to make the relationship between the key and the ciphertext as complex as possible
  - **Diffusion** is to spread the influence of a single plaintext bit over multiple ciphertext bits
- Confusion and diffusion are typically implemented with permutations and substitutions
- Symmetric encryption systems that combine permutations and substitutions in possibly multiple rounds are called **substitution-permutation ciphers** (e.g., DES)
- Substitution-permutation ciphers may be vulnerable and susceptable to brute-force attacks

Symmetric Encryption Systems

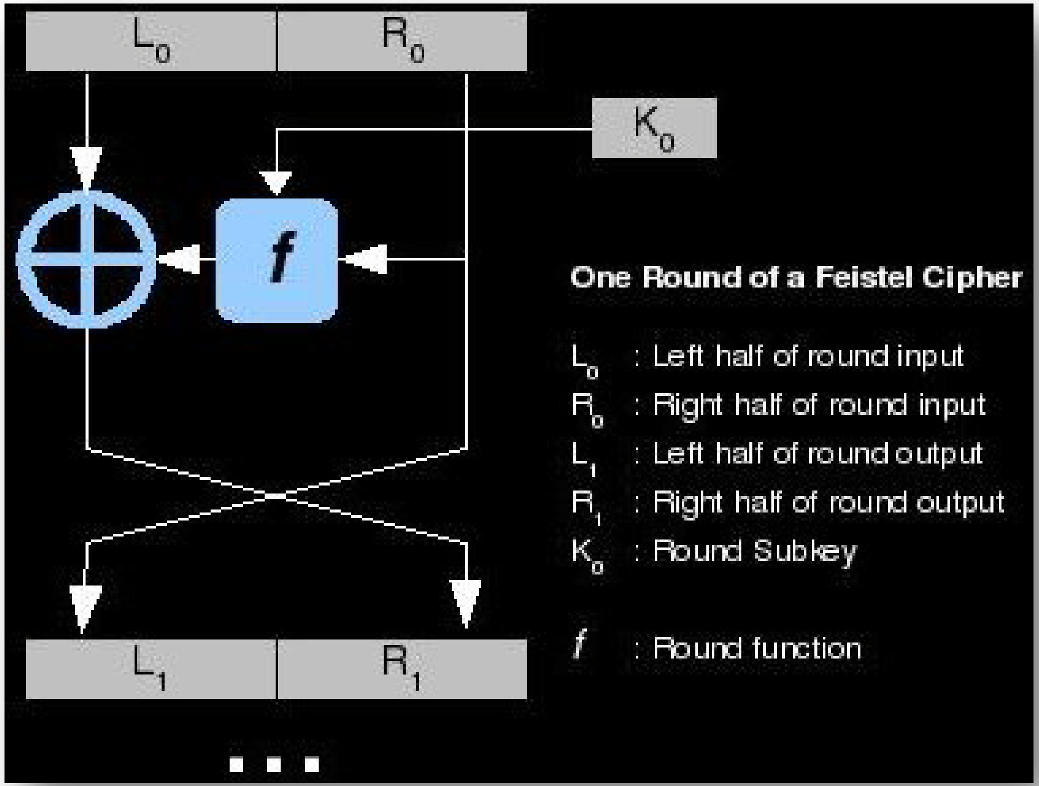
Block Ciphers – DES

- The Lucifer cipher was developed by IBM in the early 1970s
- With some modifications, the Lucifer cipher was adopted by the National Bureau of Standards (NBS) as the DES (FIPS PUB 46) in 1977
- The NIST (former NBS) reaffirmed the standard in 1983, 1988, 1993, and 1999
- FIPS PUB 46-3 (1999) also specifies the Triple Data Encryption Algrotihm (TDEA)
- The DES was officially withdrawn as a standard in 2004

Symmetric Encryption Systems

Block Ciphers – DES

- Structurally, DES is a substitution-permutation cipher that also represents a Feistel cipher (aka Feistel network)
  - $\Sigma = \mathbf{Z}_2 = \{0,1\}$
  - Block length 2t
  - r rounds
  - For every $k \in K$, r round keys $k_1,\ldots,k_r$ must be generated (represen-ting the key schedule) and used on a per-round basis


L_0
R_0
K_0
f
One Round of a Feistel Cipher
L_0 : Left half of round input
R_0 : Right half of round input
L_1 : Left half of round output
R_1 : Right half of round output
K_0 : Round Subkey
f : Round function
L_1
R_1
. . .

Symmetric Encryption Systems

Block Ciphers – DES

- The encryption function E_k starts by splitting the plaintext message block m into two halves of t bits each, i.e., $m = (L_0,R_0)$
- A sequence of pairs (L_i,R_i) for i=1,…,r is then recursively computed

$$(L_i,R_i) = (R_{i-1},L_{i-1} \oplus f_{k_i}(R_{i-1})) \qquad (1)$$

- (L_r,R_r) in reverse order represents the ciphertext block,i.e., $c=(R_r,L_r)$
- Hence, $c = E_k(m) = E_k(L_0,R_0) = (R_r,L_r)$
- Formula (1) can be written as $(L_{i-1},R_{i-1}) = (R_i \oplus f_{k_i}(L_i),L_i)$
- It is therefore possible to recursively compute L_{i-1} and R_{i-1} from L_i, R_i, and k_i, and hence to determine $m = (L_0,R_0)$ from $c = (R_r,L_r)$ using the round keys in reverse order $k_r,…,k_1$
- DES is a Feistel cipher with block length 2t = 64 bits and r = 16 rounds

Symmetric Encryption Systems

Block Ciphers – DES

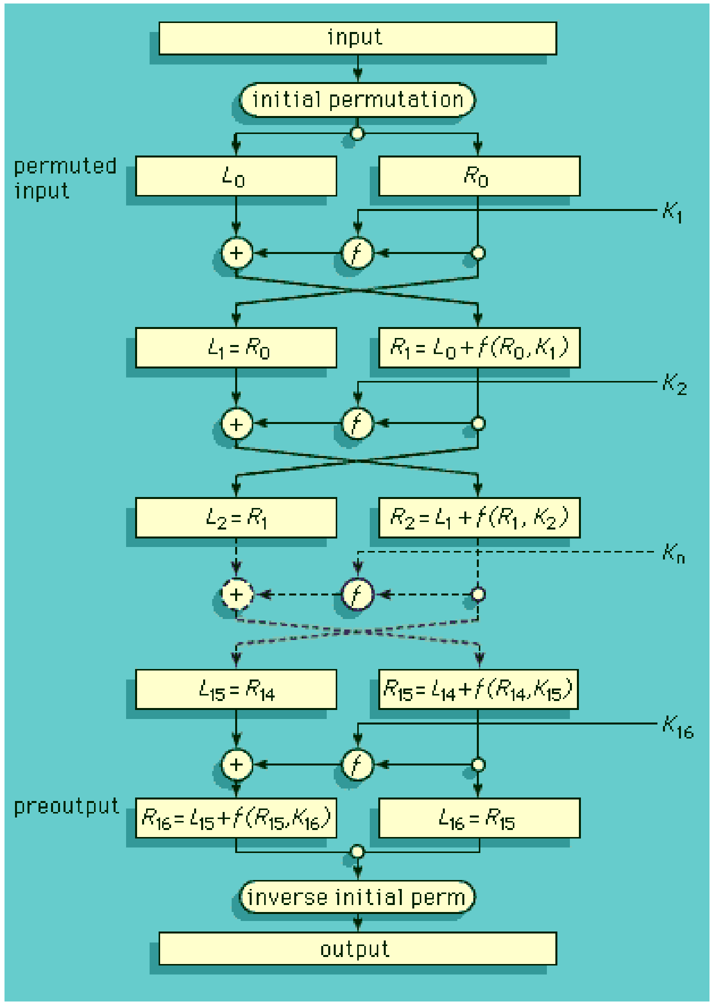
- DES keys are 64-bit strings with odd parity (where each byte is considered individually), i.e., $K = \{(k_1,\ldots,k_{64}) \in \{0,1\}^{64} \mid \Sigma_{j=1,\ldots,8}\, k_{8i+j} \equiv 1 \pmod 2 \text{ for } i = 0,\ldots,7\}$
- For example, F1DFBC9B79573413 is a valid DES key
  - `F1 = 1111000`**`1`**
  - `DF = 1101111`**`1`**
  - `BC = 1011110`**`0`**
  - `9B = 0101101`**`1`**
  - `79 = 0111010`**`1`**
  - `57 = 0101011`**`1`**
  - `34 = 0011010`**`0`**
  - `13 = 0001001`**`1`**

Symmetric Encryption Systems

Block Ciphers – DES

- To encrypt plaintext message block m using key k, the DES encryption algorithm operates in 3 steps

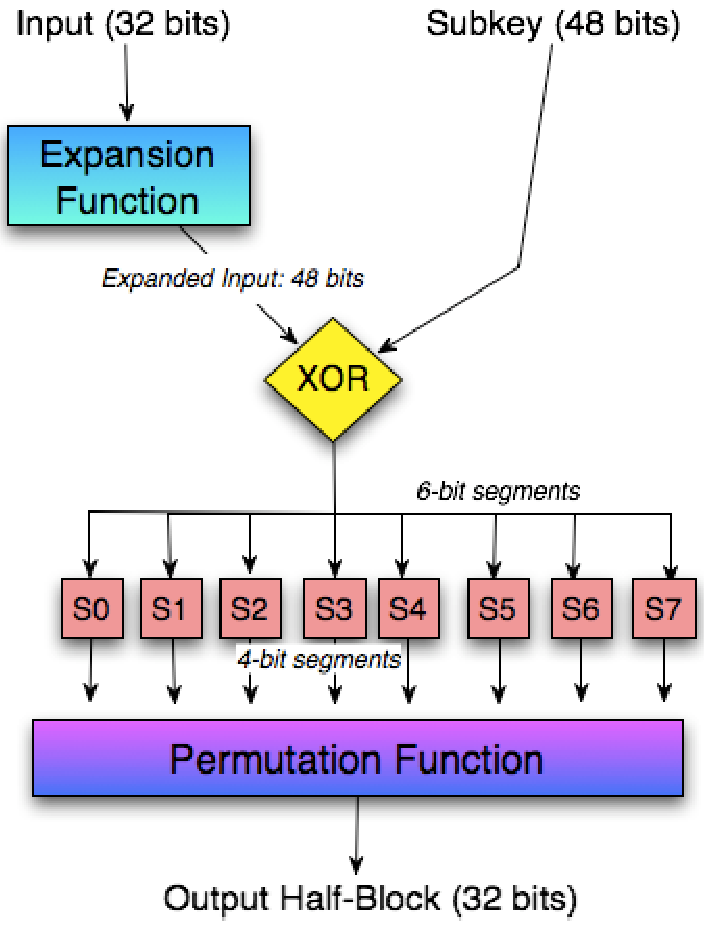
  1. An initial permutation (IP) is applied to m
  2. A 16-round Feistel cipher is applied to IP(m) – this includes the final permutation of L_{16} and R_{16}
  3. The inverse initial permutation IP^{-1} is applied to the result of step 2



Symmetric Encryption Systems

Block Ciphers – DES

- DES uses a round function f that is iterated 16 times
- It takes as input a 32-bit R-Block and a 48-bit round key k_i
- It generates as output a 32-bit value

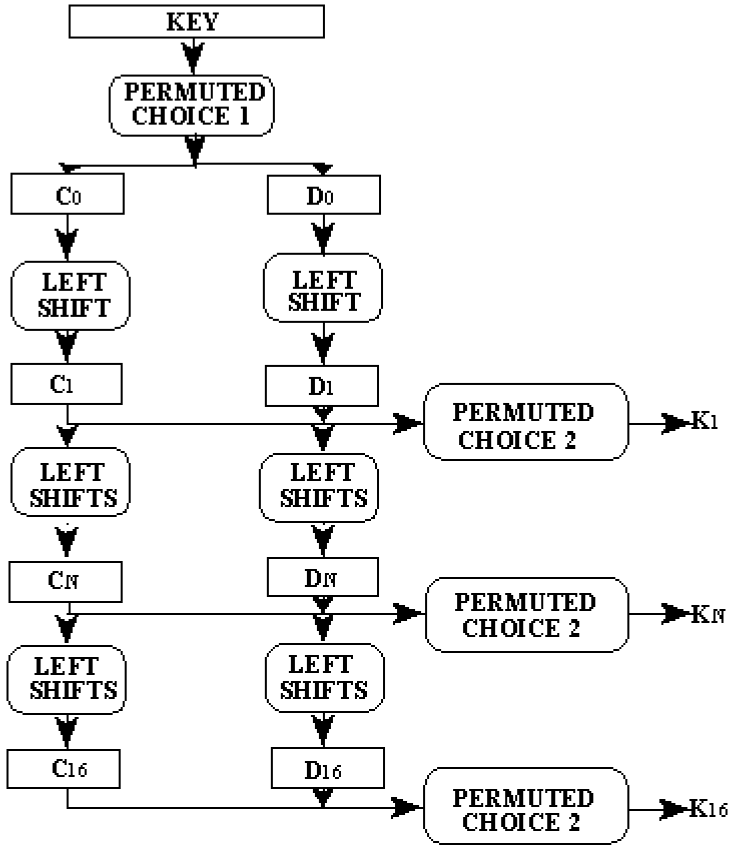


Symmetric Encryption Systems

Block Ciphers – DES

- The DES key schedule takes a 56- or 64-bit DES key and generates 16 48-bit round keys $k_1,k_2,\dots,k_{16}$
- Permuted choice (PC) functions
  - PC1: $\{0,1\}^{64} \rightarrow \{0,1\}^{28} \times \{0,1\}^{28}$
  - PC2: $\{0,1\}^{28} \times \{0,1\}^{28} \rightarrow \{0,1\}^{64}$

LEFT SHIFT
1 position in iteration 1, 2, 9, 16
2 positions in iteration 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15



Symmetric Encryption Systems

Block Ciphers – DES

- Exercise 3-4: DES
  1. Work through the CrypTool animation of the DES (CrypTool > Indiv. Procedures > Visualization of Algorithms > DES…)
  2. Discuss the differences between the DES encryption and decryption algorithms
  3. Discuss the efficiency of hardware and software implementations of the DES encryption and decryption algorithms

Symmetric Encryption Systems

Block Ciphers – DES

- Since its standardization, the DES has been subject to a lot of public scrutiny
- There are 4 weak and 12 semiweak DES keys
  - The DES key k is **weak** if $DES_k(DES_k(m)) = m$ $(\forall m \in \mathcal{M})$
  - The DES keys k_1 and k_2 are **semiweak** if $DES_{k_1}(DES_{k_2}(m)) = m$ $(\forall m \in \mathcal{M})$
- It is sometimes recommended to avoid weak and semiweak keys
- There are $16 = 2^4$ such keys, so the probability of randomly generating one is only $2^4/2^{56} = 2^{-52} \approx 2.22 \cdot 10^{-16}$

Symmetric Encryption Systems

Block Ciphers – DES

- In theory, there are a few cryptanalytical attacks against DES
  - **Differential cryptanalysis** (Biham and Shamir, late 1980s) requires 2^{47} chosen plaintexts

  - **Linear cryptanalysis** (Matsui, 1993) requires 2^{43} known plaintexts

- The designers of DES claimed that they had known differential cryptanalysis prior to its publication, and that providing protection against it had actually been a design goal

Symmetric Encryption Systems

Block Ciphers – DES

- In practice, both cryptanalytical attacks are pointless (because the amount of chosen or known plaintext is so huge)
- But the attacks are theoretically interesting and yield principles and criteria for the design of block ciphers
- All new block ciphers are routinely shown to be resistant against differential and linear cryptanalysis
- From a practical viewpoint, the major weakness and vulnerability of DES remains its relatively small key length (56 bits) and key space (2^{56} elements)
- An exhaustive key search is successful after
  - 2^{56} tries in the worst case
  - $2^{56}/2 = 2^{55}$ tries in the average case

Symmetric Encryption Systems

Block Ciphers – DES

- The complementation property

$$DES_k(m) = c \Leftrightarrow DES_{\overline{k}}(\overline{m}) = \overline{c}, \text{ i.e., } DES_k(m) = \overline{DES_{\overline{k}}(\overline{m})}$$

  can be used in a known-plaintext attack to additionally halve the key space (to 2^{54})
- Let the adversary know 2 plaintext-ciphertext pairs
  - (m,c_1) with $c_1 = DES_k(m)$
  - $(\overline{m},c_2)$ with $c_2 = DES_k(\overline{m})$
- For every key candidate k‘, he or she can compute $c = DES_{k'}(m)$
  - If $c = c_1$, then k‘ is the correct key
  - If $c = \overline{c_2}$, then $\overline{k'}$ is the correct key

Symmetric Encryption Systems

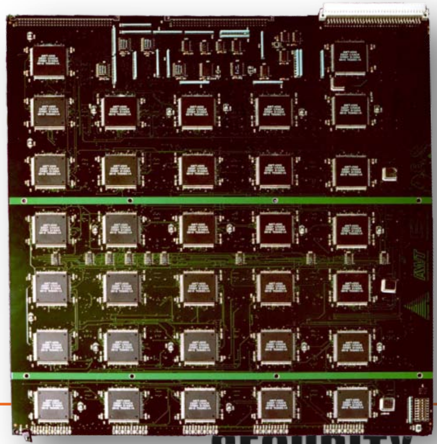
Block Ciphers – DES

- The feasibility of an exhaustive key search has been discussed since the standardization of DES in 1977
- An exhaustive key search needs a lot of time but almost no memory (simply try out all possible keys)
- On the other hand, if one has a lot of memory and is willing to precompute the ciphertext c for a known plaintext message m and all possible keys $k \in K$, then one can do an exhaustive key search almost instan-taneously (i.e., retrieve k from a table with (c,k)-entries)
- There are time-memory tradeoffs

Symmetric Encryption Systems

Block Ciphers – DES

- Many researchers have discussed the possibility to design and build a dedicated machine to do an exhaustive key search against DES
  - In 1977, Diffie and Hellman estimated that a 20-million-USD-machine could find a key in one day
  - In 1993, Michael J. Wiener estimated that a1-million-USD-machine could find a key in 3.5 hours
  - In 1997, Wiener estimated that a1-million-USD-machine could be 6 times faster (i.e., find a key in 35 minutes)
  - In 1998, the Electronic Frontier Foundation (EFF) built a massively parallel DES key search machine named Deep Crack for USD 250,000 (the machine found a DES key within 56 hours)

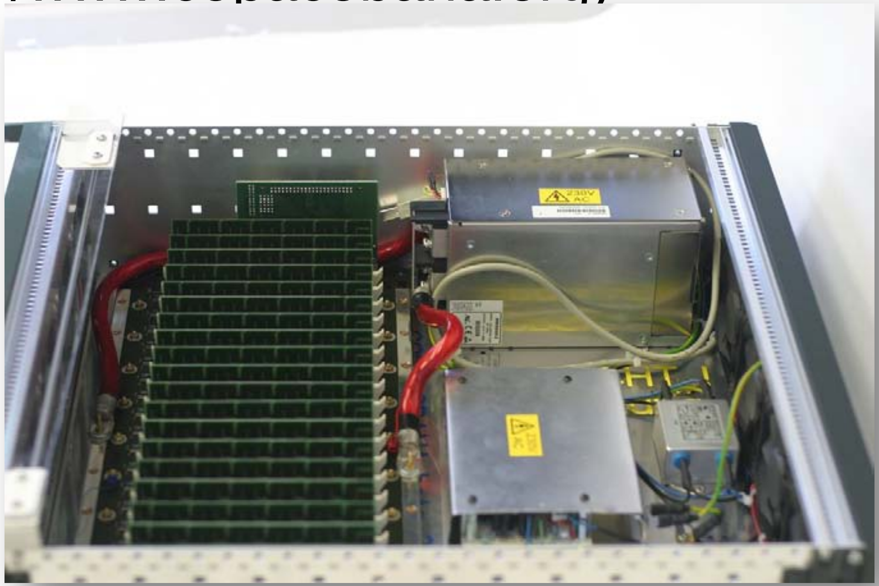


Symmetric Encryption Systems

Block Ciphers – DES

- In 2006, a group of German researchers (Ruhr University of Bochum and University of Kiel) built a DES key searching machine named **Cost-Optimized Pralallel Code Breaker (COPACOBANA)** for less than USD 10,000

- The machine found a DES key in 8.7 days
- It has been further improved (http://www.copacobana.org)



Symmetric Encryption Systems

Block Ciphers – DES

- More interestingly, one can spend idle computing cycles of net-worked computer systems to mount an exhaustive key search
- In 1991, Jean-Jacques Quisquater and Yvo Desmedt coined the term **Chinese lottery** to refer to this idea (documented in RFC 3607)

- If enough (networked) computer systems participate in an exhaus-tive key search, then a DES key can be found without dedicated machines like Deep Crack or COPACOBANA (e.g., M4 Project to break 3 original Enigma-encrypted ciphertexts)
- In 1999, the participants of the Distributed.Net project (> 100,000 computer systems) broke a DES key in only 23 hours
- Note that the participants of a key search need not be aware of this fact (due to active content)

Symmetric Encryption Systems

Block Ciphers – DES

- If one goes to key lengths beyond 100 bits (e.g., 128 bits), then the resulting ciphers are resistant against exhaustive key search
- If one considers quantum computers, then one has to increase the key length to 200 bits and more (→ post-quantum cryptography)
- Unfotunately, this does not mean that all ciphers with these key lengths and all respective implementations are secure under all circumstances

Symmetric Encryption Systems

Block Ciphers – DES

- In practice, there are several possibilities to address the small key length problem of DES
  - The way DES is used can be modified
    - Prevention of known plaintext
    - Complex key setup procedures
    - Frequent key changes (e.g., on a per-packet basis)
    - All-or-nothing transform (Rivest)
  - DES can be modified in a way that compensates for ist relatively small key length (e.g., DESX)
  - DES can be iterated multiple times (e.g., TDEA)
  - An alternative symmetric encryption system with a larger key length may be used (e.g., AES)

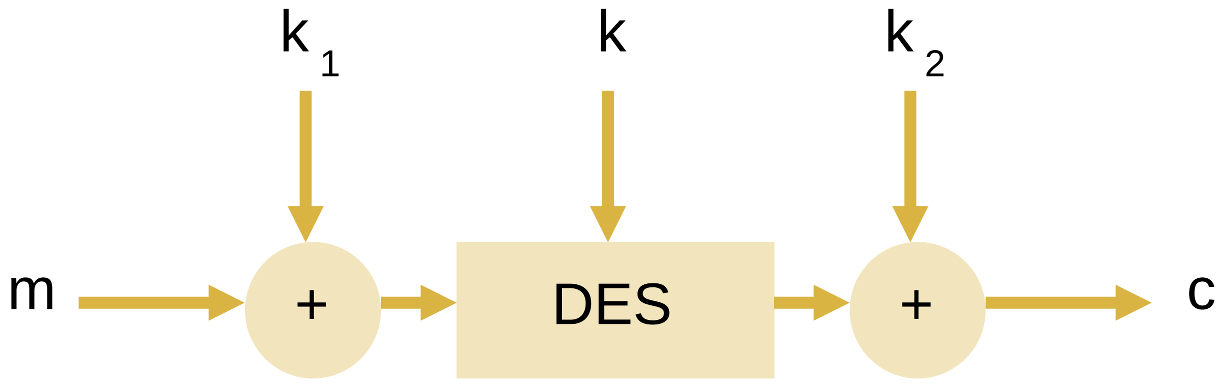
Symmetric Encryption Systems

Block Ciphers – DESX

- Rivest proposed to apply Merkle‘s key whitening technique to protect DES (only) against exhaustive key search

$$c = DESX_{k_1,k,k_2}(m) = k_2 \oplus DES_k(m \oplus k_1)$$

- DESX requires 56 + 64 + 64 = 184 bits of keying material
- It is employed in Microsoft‘s Encrypted File System (EFS)



Symmetric Encryption Systems

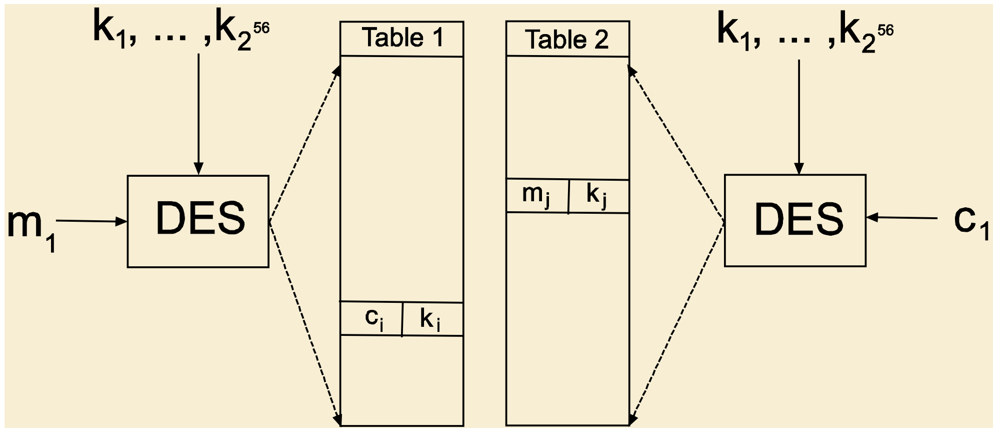
Block Ciphers – TDEA

- Preliminary remarks regarding multiple iterations of a block cipher
  - Multiple iterations should be done with different keys
  - Multiple iterations with different keys only provide a security advantage, if the encryption functions are not closed with regard to concatenation, i.e., they do not form a group (otherwise $E_{k_3}(m) = E_{k_2}(E_{k_1}(m))$)
- It was shown in the 1990s (by Wiener) that the DES encryption functions do not form a group, and hence that multiple iterations of DES may provide a security advantage
- The first (meaningful) possibility to iterate DES would be a double encryption with two different (and independent) keys
- But double DES is susceptible to a **meet-in-the-middle attack**

Symmetric Encryption Systems

Block Ciphers – TDEA

- An adversary who knows a few plaintext-ciphertext pairs (m_i,c_i) – where c_i is derived from a double encryption of m_i using k_1 and k_2 – can mount a meet-in-the-middle attack
- He starts with (m_1,c_1) and builds 2 large tables (256 entries each)



- If $c_i = m_j$, then (k_i,k_j) is a possible key pair that can be verified with (m_2,c_2) or another plaintext-cipher-text-pair
- The mere existence of the attack is reason enough to use TDEA

Symmetric Encryption Systems

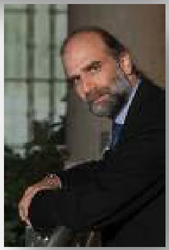
Block Ciphers – TDEA

- A DEA key consists of a key bundle $k = (k_1,k_2,k_3)$
- FIPS PUB 46-3 specifies several options for the key bundle
- Encryption function $c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$
- Decryption function $m = D_{k_1}(E_{k_2}(D_{k_3}(m)))$
- Iterating a block cipher multiple times can be done with any block cipher and there is nothing DES-specific about the TDEA construction
- The main disadvantage of TDEA is performance

Symmetric Encryption Systems

Block Ciphers – AES

- From 1997 to 2000, the U.S. NIST carried out an open competition with the aim to standardize an **Advanced Encryption Standard (AES)** as the successor of the DES
- Contrary to the DES standardization effort, many parties from industry and academia participated in the AES competition
- 5 finalists
  - MARS (Coppersmith et al.)
  - RC6 (Rivest)
  - Rijndael (Daemen and Rijmen)
  - Serpent (Anderson, Biham, and Knudsen)
  - Twofish (Schneier et al.)



- In 2000, the NIST proposed Rijndael as AES in FIPS PUB 197

Symmetric Encryption Systems

Block Ciphers – AES

- Official AES versions

	N_b	N_k	N_r
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

N_b = Block length (in 32-bit words) - 128 bits

N_k = Key length (in 32-bit words) - 128, 192, or 256 bits

N_r = Number of rounds - 10, 12, or 14 rounds

- The AES is byte-oriented
- A byte can be represented differently
  - Binary representation
  - Hexadecimal representation
  - Polynomial representation (i.e., the bits refer to the coefficients of the polynomial $b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0=\Sigma_{i=0,\ldots,7}\, b_ix^i$ over $\mathbf{Z}_2$)

10100011

0xA3

x^7+x^5+x+1

Symmetric Encryption Systems

Block Ciphers – AES

- With regard to the AES, the polynomial representation is needed (mainly for byte multiplication)
- To make sure that the degree of a resulting polynomial is not greater than 7, one must take the result modulo an irreducible polynomial of degree 8
- In the case of AES, this polynomial is $f(x) = x^8+x^4+x^3+x+1$
- Because $f(x)$ is an irreducible polynomial over $\mathbf{Z}_2$, $\mathbf{Z}_2[x]_f$ is actually a field
- This field is isomorph to $GF(2^8)$, and hence this field is also known as the **AES field**

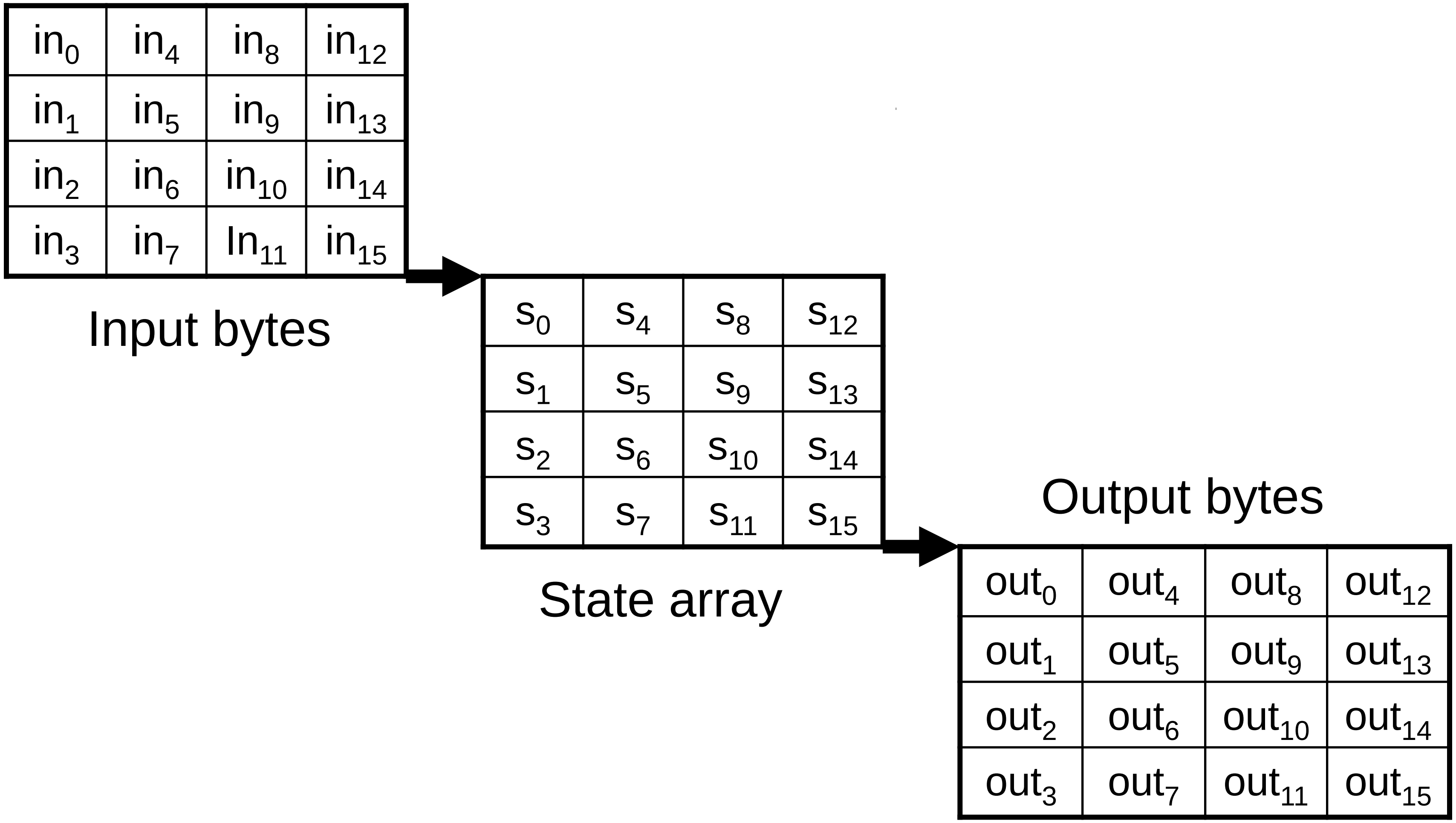
Symmetric Encryption Systems

Block Ciphers – AES

- Internally, the the AES operates on a two-dimensional array (i.e., matrice) s of bytes (aka State or State array)
- More specifically, s consists of
  - 4 rows
  - 4 columns ($N_b = 4$ for all official versions of the AES)
- Consequently, the State represents a (4x4)-matrice and each entry $s_{r,c}$ or s[r,c] ($0 \leq r,c \leq 4$) comprises one byte

Symmetric Encryption Systems

Block Ciphers – AES



in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 In_{11} in_{15}
Input bytes
s_0 s_4 s_8 s_{12}
s_1 s_5 s_9 s_{13}
s_2 s_6 s_{10} s_{14}
s_3 s_7 s_{11} s_{15}
State array
Output bytes
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

Symmetric Encryption Systems

Block Ciphers – AES

AES encryption algorithm

```
(in)
------------------------------------------------
s ← in
s ← AddRoundKey(s,w[0,$N_b$-1])
for r = 1 to ($N_r$-1) do
    s ← SubBytes(s)
    s ← ShiftRows(s)
    s ← MixColumns(s)
    s ← AddRoundKey(s,w[r$N_b$,(r+1)$N_b$-1])
s ← SubBytes(s)
s ← ShiftRows(s)
s ← AddRoundKey(s,w[$N_r N_b$,($N_r$+1)$N_b$-1])
out ← s
------------------------------------------------
(out)
```

Symmetric Encryption Systems

Block Ciphers – AES

- The **SubBytes()** transformation implements a nonlinear substitution cipher, i.e., each byte $s_{r,c}$ is substituted with another byte $s'_{r,c}$
- The substitution is specified in a substitution table (aka **S-box**)
- The S-box is a (16x16)-matrix of bytes
- Contrary to the DES S-box, the AES S-box is generated in a mathematically structured and well-documented way

Symmetric Encryption Systems

Block Ciphers – AES

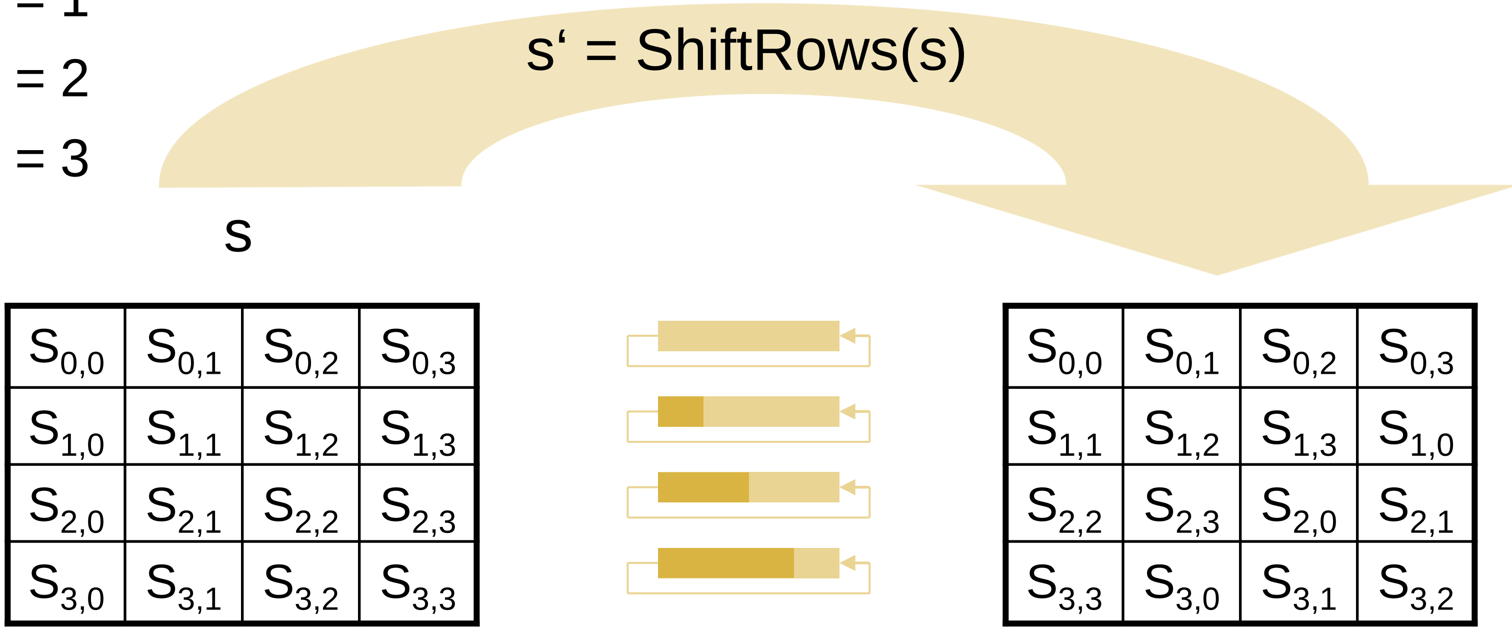
```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76
10 |ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
20 |b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
30 |04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75
40 |09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84
50 |53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
60 |d0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8
70 |51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2
80 |cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73
90 |60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db
a0 |e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79
b0 |e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08
c0 |ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a
d0 |70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e
e0 |e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df
f0 |8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16
```

AES S-box

Symmetric Encryption Systems

Block Ciphers – AES

- The **ShiftRows()** transformation implements a cyclic shift left (rotation) for the bytes in row r ($0 \leq r \leq 3$) for r positions
- $s'_{r,c} = s_{r,c+shift(r,N_b) \bmod N_b}$
  - shift(0,4) = 0
  - shift(1,4) = 1
  - shift(2,4) = 2
  - shift(3,4) = 3



$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Symmetric Encryption Systems

Block Ciphers – AES

- The **MixColumns()** transformation implements an invertible linear transformation for each column
- More specifically, each column c ($0 \leq c \leq 3$) is treated as a four-term polynomial $s_c(x)$ over GF(2^8)

$$s_c(x) = s_{3,c}x^3 + s_{2,c}x^2 + s_{1,c}x + s_{0,c}$$

  and multiplied modulo ($x^4 + 1$) with the fixed polynomial

$$c(x) = \{\texttt{03}\}x^3 + \{\texttt{01}\}x^2 + \{\texttt{01}\}x + \{\texttt{02}\}$$

- Consequently, $s'_c(x) = s_c(x)c(x) \pmod{(x^4 + 1)}$

Symmetric Encryption Systems

Block Ciphers – AES

- Alternatively, the **MixColumns()** transformation transformation can be written as a matrix multiplication (with a specially crafted operator ⊗)

$$s'(x) = c(x) \otimes s(x)$$

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix}$$

Symmetric Encryption Systems

Block Ciphers – AES

- In the **AddRoundKey()** transformation, a word of the key schedule w is added modulo 2 to each column of the State
- For $0 \leq c < N_b$ and $0 \leq r \leq N_r$,

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus w[rN_b + c]$$

- Because the AddRoundKey() transformation only comprises of a bitwise addition modulo 2, it is its own inverse

Symmetric Encryption Systems

Block Ciphers – AES

- The AES **key expansion algorithm** takes a key k and generates a key schedule w
- The key schedule, in turn, is $N_b(N_r+1)$ words long (an inital set of N_b words and additional N_b words for each N_r rounds)
- Hence, w is actually an $N_b(N_r+1)$-long array of 4-byte words

Symmetric Encryption Systems

Block Ciphers – AES

AES decryption algorithm

```
(in)
s ← in
s ← AddRoundKey(s,w[$N_rN_b$,($N_r$+1)$N_b$-1])
for r = $N_r$-1 downto 1 do
    s ← InvShiftRows(s)
    s ← InvSubBytes(s)
    s ← AddRoundKey(s,w[r$N_b$,(r+1)$N_b$-1])
    s ← InvMixColumns(s)
s ← InvShiftRows(s)
s ← InvSubBytes(s)
s ← AddRoundKey(s,w[0,$N_b$-1])
out ← s
(out)
```

Symmetric Encryption Systems

Block Ciphers – AES

- Exercise 3-5: AES

  1. Work through the CrypTool animation of the AES (CrypTool > Indiv. Procedures > Visualization of Algorithms > AES…)
  2. Discuss the efficiency of hardware and software implementations of the AES encryption and decryption algorithms

Symmetric Encryption Systems

Block Ciphers – AES

- In June 2003, the NSA announced that the AES may be used for the encryption of classified information
  - *The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths.*
- This announcement is remarkable, because it is the first time the public has access to a cipher for TOP SECRET information
- Due to the fact that the implemention is key to the security of a cryptographic system, the NSA further announced that
  - *The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use.*

Symmetric Encryption Systems

Block Ciphers – AES

- Outside the NSA, the AES has been subject to a lot of public scrutiny (especially since its standardization in FIPS PUB 197)
  - The good news is that the AES is designed in a way that it is resistant against all known cryptanalytical attacks (e.g., differential and linear cryptanalysis)
  - The bad news is that new cryptanalytical techniques that exploit the mathemtical structure of the AES have also been developed (and will continue to be refined) to eventually break the AES
- The AES and reduced-round versions thereof are popular targets for cryptanalytical attacks

Symmetric Encryption Systems

Block Ciphers – AES

- More recently, related key attacks have been successfully moun-ted against AES-192 and AES-256 (surprisingly, AES-128 has remained unaffected)
  - The full-round versions can be broken with a time complexity of 2^{176} (AES-192) or 2^{119} (AES-256)
  - The reduced-round versions can be broken with a time complexity of 2^{39} (AES-256 with 9 rounds), 2^{45} (AES-256 with 10 rounds), or 2^{70} (AES-256 with 11 rounds)
- Resistance may be improved by increasing the number of rounds, e.g., AES-128: 10 $\rightarrow$ 16, AES-192: 12 $\rightarrow$ 20, and AES-256:14 $\rightarrow$ 28
- Future releases of the standard will likely take this into account

Symmetric Encryption Systems

Block Ciphers – Modes of operation

- Confidentiality modes (NIST SP 800-38A, → http://csrc.nist.gov/groups/ST/toolkit/BCM/)
  - Electronic code book (ECB)
  - Cipherblock chaining (CBC)
  - Cipher feedback (CFB)
  - Output feedback (OFB)
  - Counter (CTR)

  } Modes of operation that effectively turn a block cipher into a stream cipher (i.e., the block cipher is used to generate a sequence of pseudorandom bits that are added modulo 2 to the plaintext)
- Authentication mode (NIST SP 800-38B)
  - Cipher-based MAC (CMAC)
- Authenticated encryption mode (NIST SP 800-38C)
  - Counter with CBC-MAC (CCM) mode

Symmetric Encryption Systems

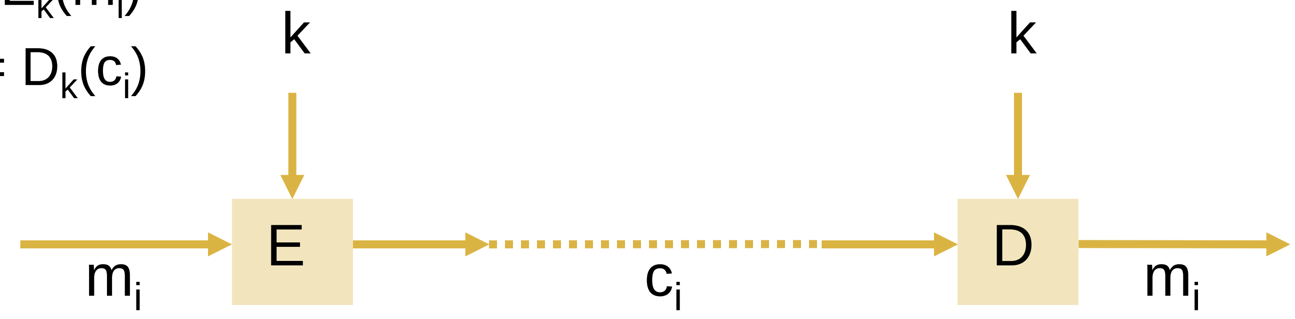
Block Ciphers – Modes of operation

- High-throughput authenticated encryption mode (→ NIST SP 800-38D)
  - Galois/counter mode (GCM)
- Future modes (work in progress)
  - AES Key Wrap (AESKW)

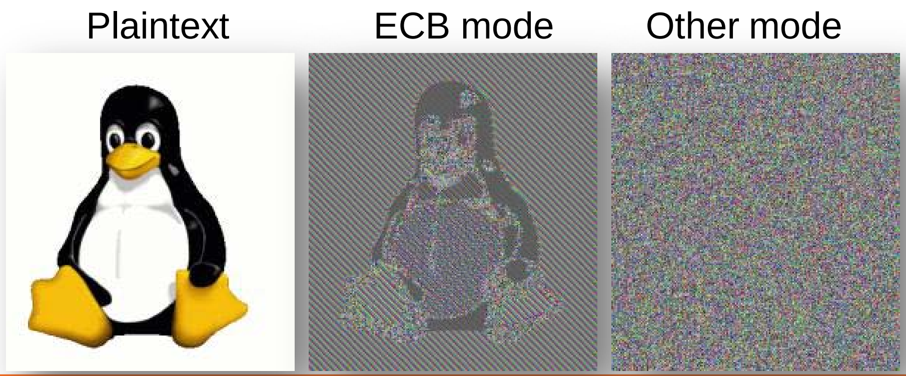
Symmetric Encryption Systems

Block Ciphers – Modes of operation

- ECB mode
  - Encryption: $c_i = E_k(m_i)$
  - Decryption: $m_i = D_k(c_i)$



- The ECB mode of operation is inappropriate for the encryption of highly structured data

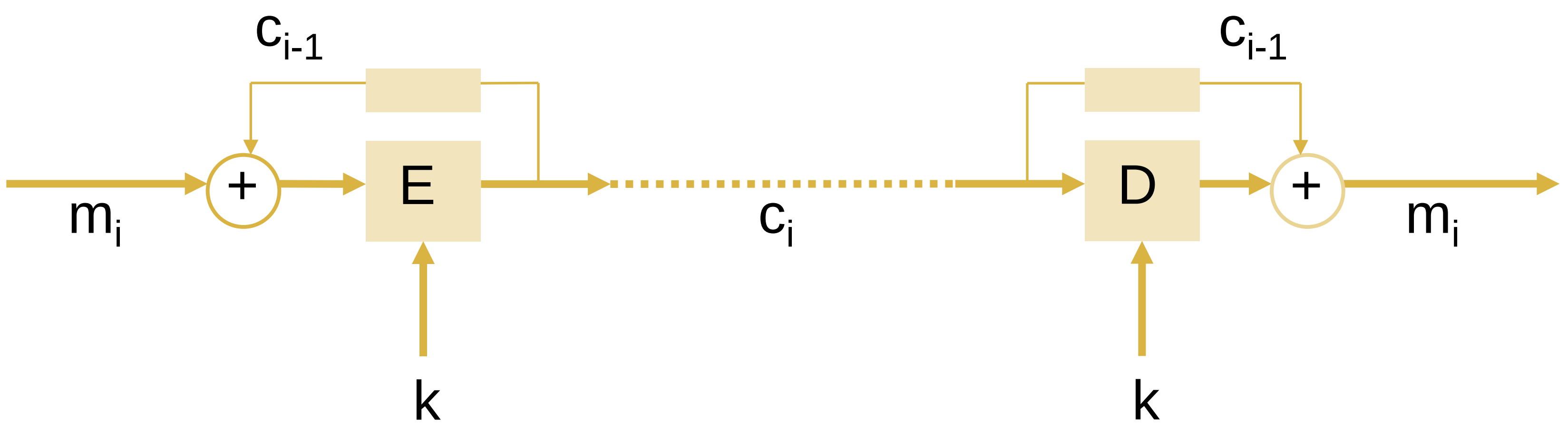


© http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Symmetric Encryption Systems

Block Ciphers – Modes of operation

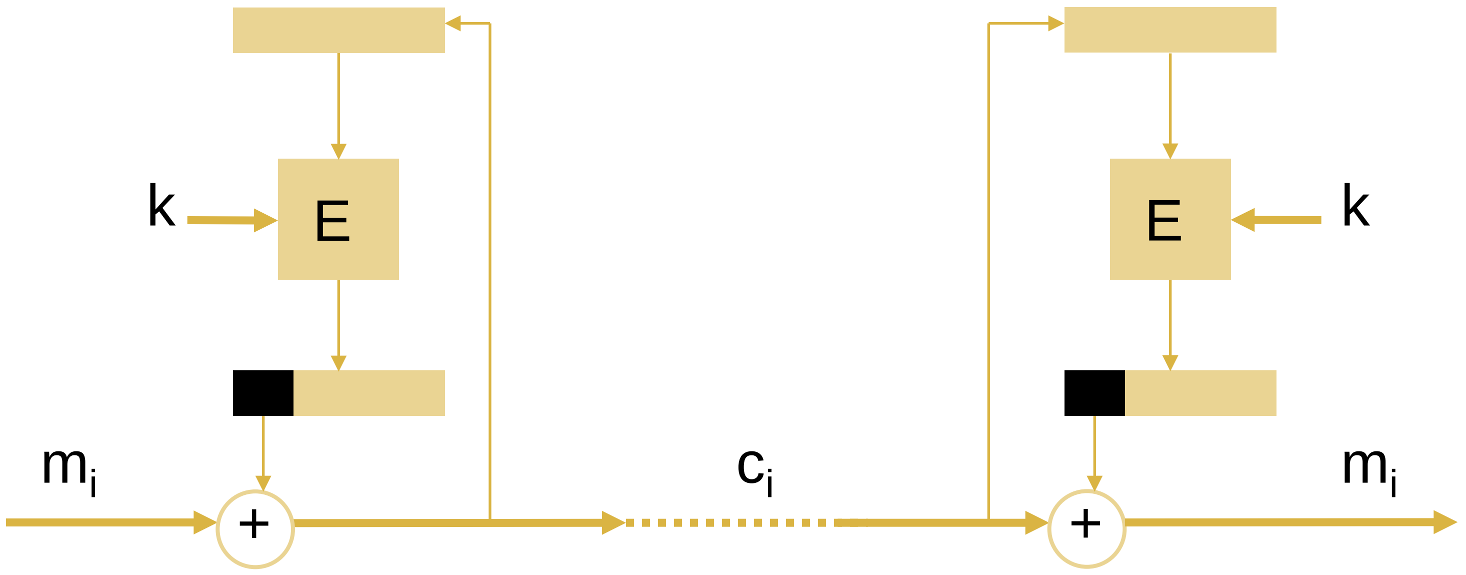
- CBC mode
  - Encryption: $c_0 = IV$ and $c_i = E_k(m_i \oplus c_{i-1})$ for i>0
  - Decryption: $c_0 = IV$ and $m_i = D_k(c_i) \oplus c_{i-1}$ for i>0



Symmetric Encryption Systems

Block Ciphers – Modes of operation

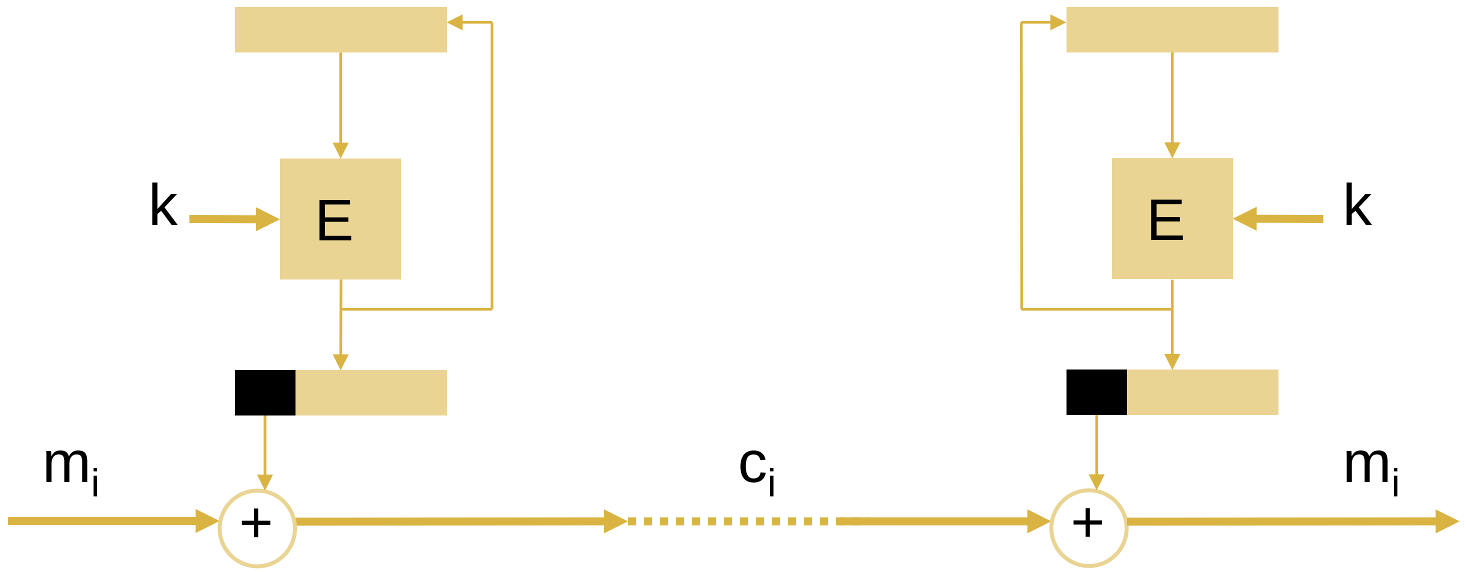
- CFB mode – nonsychronous (self-synchronizing) stream cipher



Symmetric Encryption Systems

Block Ciphers – Modes of operation

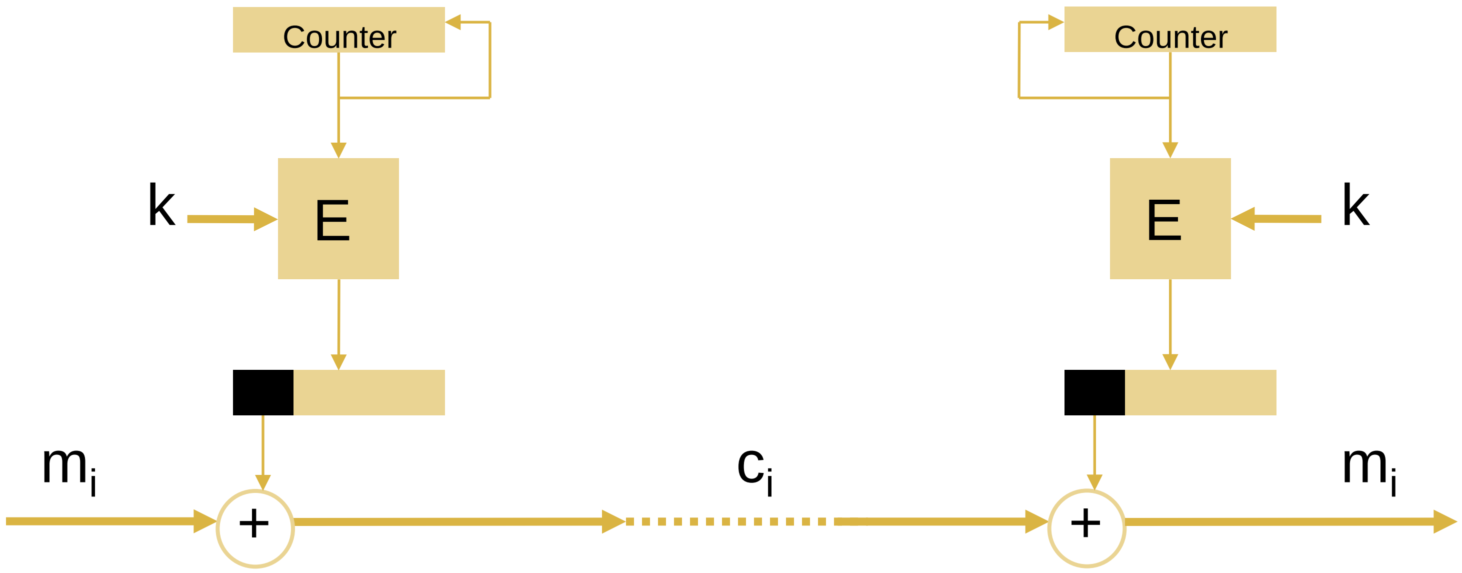
- OFB mode – sychronous (additive) stream cipher



Symmetric Encryption Systems

Block Ciphers – Modes of operation

- In CTR mode, the block cipher is used to iteratively encrypt a counter
- The ciphertexts represent the key stream
- The key stream is then added modulo 2 to the plaintext message



Symmetric Encryption Systems

3.3 Stream Ciphers

- Stream ciphers used to play an important role in cryptography
- Many (proprietary) symmetric encryption systems used in military are stream ciphers
- Stream ciphers use internal state, i.e., the i^{th} ciphertext unit depends on the i^{th} plaintext unit, the secret key, and this state
  - Synchronous or additive stream ciphers (e.g., block ciphers in OFB or CTR mode, Vernam cipher, …)
  - Nonsynchronous or self-synchronizing stream ciphers (e.g., block ciphers in CFB mode)

Symmetric Encryption Systems

Stream Ciphers

- Let $\Sigma = \mathbf{Z}_2 = \{0,1\}$, $M = C = \Sigma^*$, and $K = \Sigma^n$
- To encrypt an l-bit plaintext message $m = m_1 \ldots m_l \in M$, a secret key $k \in K$ must be expanded into an l-bit key stream $k_1 \ldots k_l$
- Encryption function (of an additive stream cipher)

$$E_k(m) = m_1 \oplus k_1, \ldots, m_l \oplus k_l = c_1, \ldots, c_l$$

- Decryption function

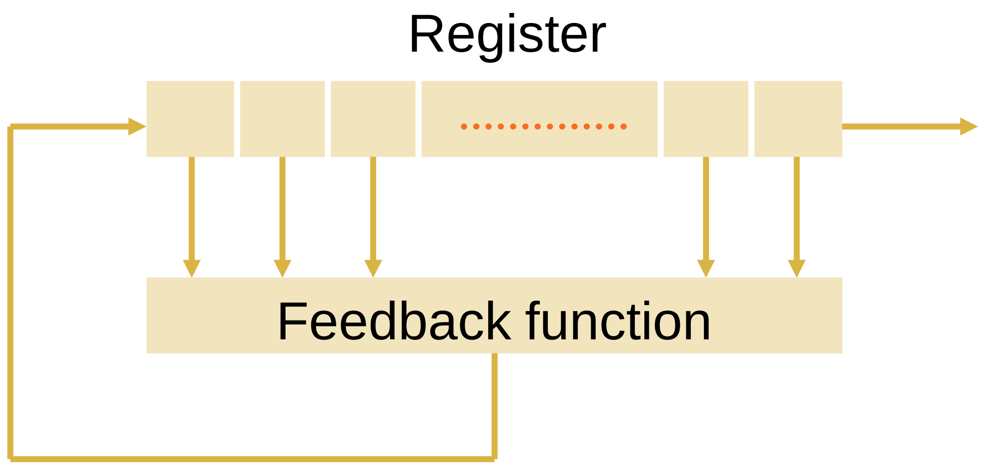
$$D_k(c) = c_1 \oplus k_1, \ldots, c_l \oplus k_l = m_1, \ldots, m_l$$

- With regard to the design of an additive stream cipher, the main question is how to expand k into a key stream $(k_i)_{i \geq 1}$
- Classes of additive stream ciphers
  - LSFR-based stream ciphers
  - Other stream ciphers

Symmetric Encryption Systems

Stream Ciphers

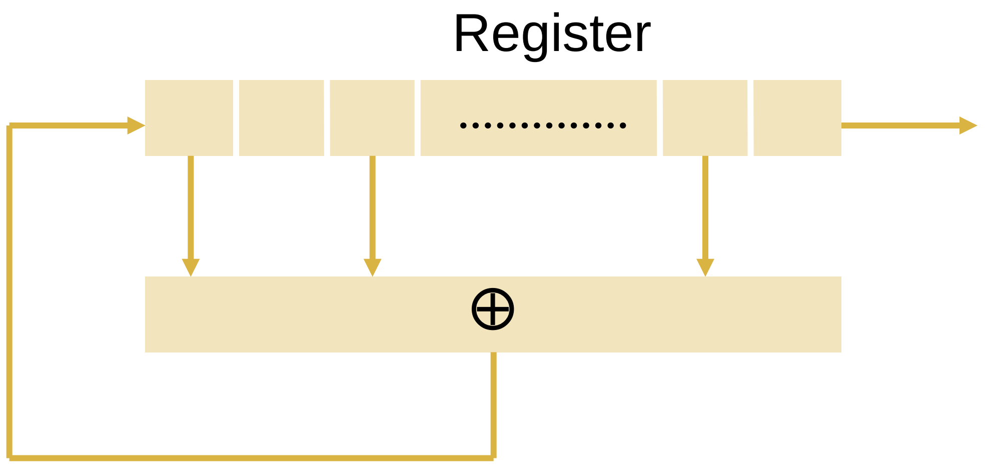
- One possibility to generate a key stream $(k_i)_{i\geq 1}$ is to use a **feedback shift register (FSR)**
- The feedback function can be arbitrary



Symmetric Encryption Systems

Stream Ciphers

- If the FSR operates in a field and the feedback function is linear over this field, then the FSR represents a **linear feedback shift register (LFSR)**
- In GF(2), the feedback function is linear iff it represents the addition modulo 2 of some register cells



Symmetric Encryption Systems

Stream Ciphers – LFSR-based Stream Ciphers

- Once the register is initialized with a seed, it operates deterministically, i.e., the sequence of the values generated is deterministic by the (current or previous) state of the register
- Because the register has a finite number of possible states, it must enter into a cycle
- LFSRs are well studied and understood
- In particular, there is a mathematical theory to choose the linear feedback function so that the cycle is as long as possible, i.e., it cycles through all 2^n-1 states for an n-bit regsiter
- LFSRs and LFSR-based stream ciphers can be efficiently implemented in hardware
- Consequently, there is room for other (preferrably additive) stream ciphers optimized for software implementations

Symmetric Encryption Systems

Stream Ciphers – Other Stream Ciphers

- **RC4** is an additive stream cipher proposed by Rivest in 1987 (trade secret of RSA Security or EMC)
- It is deployed in many products and cryptographic seciroty protocols (e.g., SSL/TLS, WEP, WPA, …)
- In 1994, the source code of an RC4 implementation was anonymously posted to the Cypherpunks mailing list
- Due to the trademark protection of RC4, this algorithm is referred to as **ARCFOUR (Alleged-RC4)**

Symmetric Encryption Systems

Stream Ciphers – Other Stream Ciphers

- RC4/ARCFOUR takes a variable-length key between 1 and 256 bytes (8…2,048 bits) and generates a sequence of pseudorandomly generated bytes
- Each key byte is then added modulo 2 to a plaintext byte
- It employs an array S of 256 bytes of State information (S-box)
- The S-box bytes are labeled S[0], S[1], … , S[255]
- The S-box is initialized in 3 steps
  1. S[i] = i for i = 0,1,…,255
  2. An auxiliary array S_2 of 256 bytes is allocated and filled with the key, repeating bytes as necessary
  3. The S-box is initialized as indicated on the right hand

```
(S)
----------------------------------------
j ← 0
for i = 0 to 255 do
    j ← (j + S[i] + S_2[i]) mod 256
    S[i] ↔ S[j]
----------------------------------------
(S)
```

Symmetric Encryption Systems

Stream Ciphers – Other Stream Ciphers

- Afterwards, the algorithm to generate a key byte k from S is relatively simple and straightforward
- It may be executed multiple times
- i and j are global variables that are initially set to 0

```
(S)
________________________________
$i \leftarrow (i + 1) \bmod 256$
$j \leftarrow (j + S[i]) \bmod 256$
$S[i] \leftrightarrow S[j]$
$t \leftarrow (S[i] + S[j]) \bmod 256$
$k \leftarrow S[t]$
________________________________
(k)
```

Symmetric Encryption Systems

Stream Ciphers – Other Stream Ciphers

- Due to its wide deployment, RC4/ARCFOUR is a popular target to attack
- The randomness properties of the first bytes of the generated keystream are known to be poor
- This vulnerability has been exploited in attacks against WEP (key exposure in less than 1 minute)
- These attacks have disturbed users of wireless networks
- The attacks can be defeated by ignoring (and not using) the first bytes

Symmetric Encryption Systems

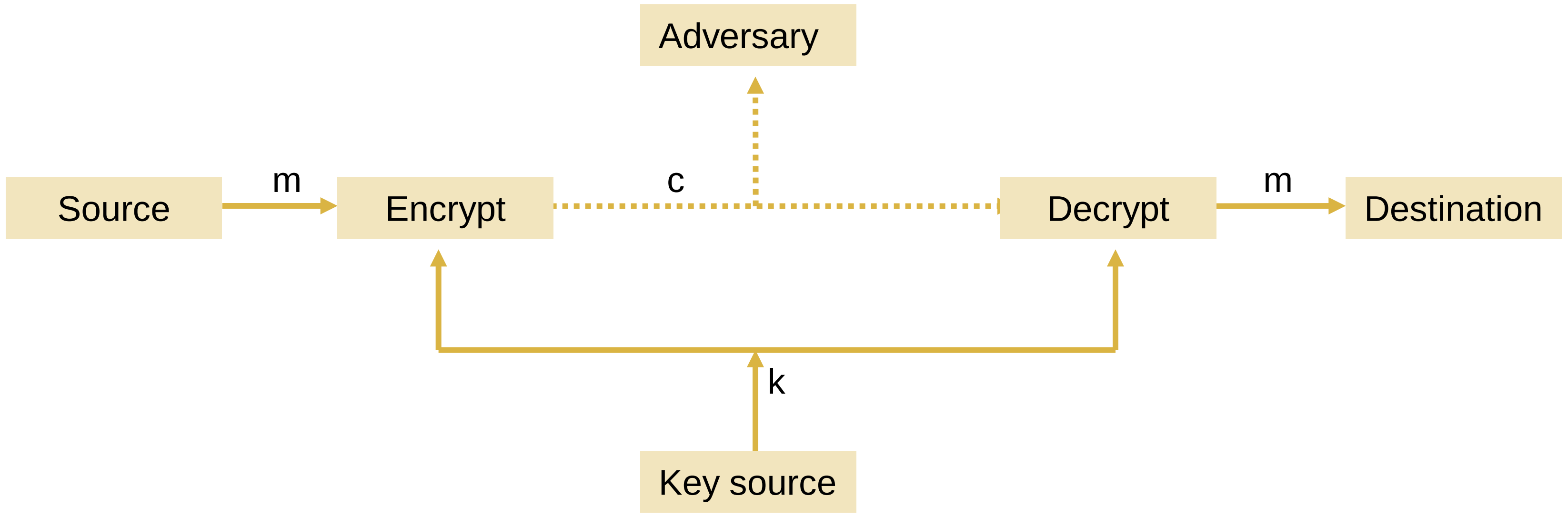
3.4 Perfectly Secure Encryption

- The field of perfectly (i.e., information-theoretically) secure encryption was pioneered by Shannon in the late 1940s
- The aim was to design an encryption system that makes it impossible (even for a very powerful and skilled adversary) to derive information about a plaintext message from a ciphertext
- It is not obvious that such an absolute notion of security can be achieved in the first place

Symmetric Encryption Systems

Perfectly Secure Encryption

- Shannon‘s model of a symmetric encryption system



- The model can be extended to comprise probabilistic or randomized encryption

Symmetric Encryption Systems

Perfectly Secure Encryption

- Formally, an encryption (process) that takes place in a symmetric encryption system (*M,C,K,E,D*) can be viewed as a discrete random experiment
- M and K represent independent real-valued random variables with probability distributions P_M: $M \rightarrow \mathbf{R}^+$ and P_K: $K \rightarrow \mathbf{R}^+$
  - P_M typically depends on the plaintext message language in use
  - P_K is uniformly distributed over all possible keys, i.e., all keys are ideally equiprobable
- In such a setting, there is a third random variable C (modelling the ciphertext) distributed according to P_C: $C \rightarrow \mathbf{R}^+$
- The probability distribution P_C is completely determined by P_M and P_K

Symmetric Encryption Systems

Perfectly Secure Encryption

- The random variable C is the one that a (passive) adversary can observe and analyze to derive information about M or K (i.e., he tries to find messages or keys that are more probable than others)
- Perfect secrecy can be achieved if the ability to observe and analyze C does not yield any additional information about M or K
- This means that observing the ciphertext does not help the adversary
- Alternatively speaking, the a posteriori probabilities of the plaintext messages (i.e., the probabilities that can be assigned after having observed the ciphertext) are equal to the respective a priori probabilities (i.e., the probabilities that can be assigned without having observed the ciphertext)

Symmetric Encryption Systems

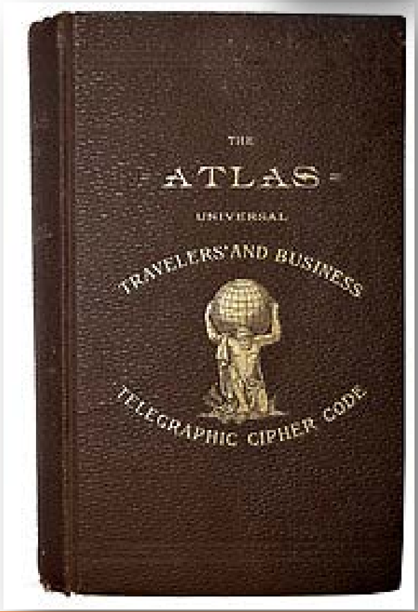
Perfectly Secure Encryption

- In information theory, the **entropy** H of a random variable measures its incertainty (i.e., the incertainty about its outcome)
- A symmetric encryption system (*M,C,K,E,D*) is **perfectly secure** if H(M|C) = H(M) for every probability distribution P_M
- Shannon showed that H(K) ≥ H(M) must hold for a perfectly secure symmetric encryption system
- Hence, the encryption key must be at least as long as the plaintext message (this is impractical)

Symmetric Encryption Systems

Perfectly Secure Encryption

- In the 1910s, Gilbert S. Vernam and Joseph O. Mauborgne proposed a simple device that implements the **one-time pad**
- Due to a U.S. patent that was granted to Vernam in 1919, the one-time pad is also known as **Vernam cipher**
- It can be proven to be perfectly secure
- In 2011, it was discovered by Steven M. Bellovin that the one-time pad had been known almost 35 years earlier to Frank Miller who published a book entitled “Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams” in 1882



Symmetric Encryption Systems

3.5 Final Remarks

- The symmetric encryption systems in use today are similar in the sense that they all employ a mixture of more or less complex functions that are iterated multiple times
- The result is something that is inherently difficult to analyze
- There are details that sometimes look mysterious or arbitrary to outsiders (e.g., S-boxes of DES)
- This sometimes misleads people to believe that they can design a new cipher on their own (this is dangerous)
- Unless one enters the field of information-theoretically secure encryption systems, the level of security (and assurance) an encryption system provides is difficult to determine and quantify
- One reason is that it is difficult to exhaustively say what cryptanalytical attacks are feasible or discovered in the future

Symmetric Encryption Systems

Final Remarks

- The bottom line is that fairly little is known about the real security of (symmetric) encryption systems used in the field
- It is therefore simple to put in place rumors about weaknesses and potential vulnerabilities of (possibly competing) systems
- The media play an important role

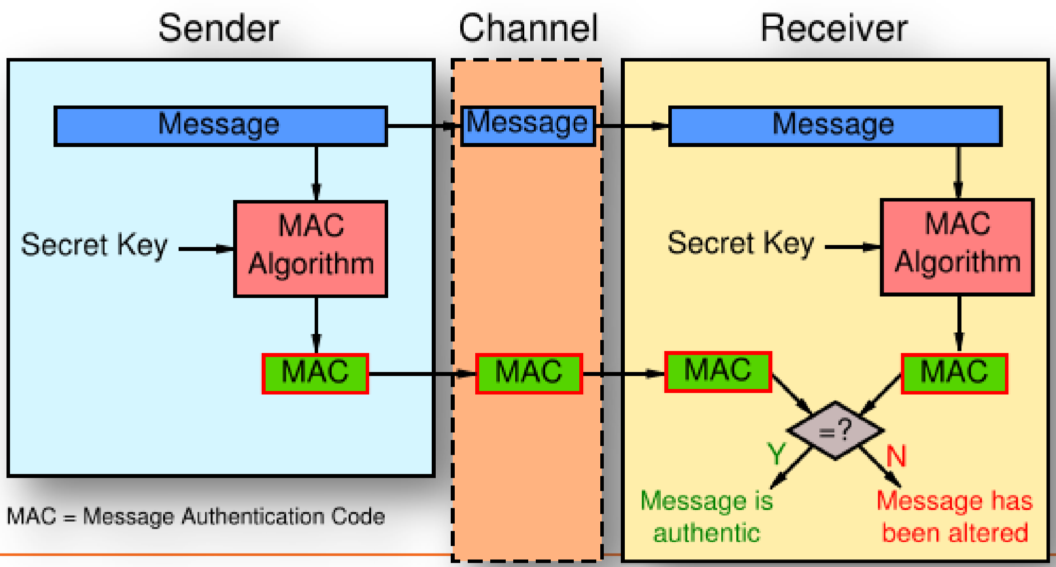
Finishing kick
Hackers hit
"What's been taken is bits of data that the
... put together into an identity."

4 Message Authentication

Message Authentication

4.1 Introduction

- Message authentication
  - Public key cryptography → Digital signatures
  - Secret key cryptography → Message authentication codes (MACs)
- A MAC is an authentication tag that is computed and verified with a secret key (it is not qualified to provide nonrepudiation services)



Message Authentication

Introduction

- A message authentication system consists of 5 components
  - Message space M
  - Tag space T
  - Key space K
  - Family $A = \{A_k : k \in K\}$ of authentication functions $A_k : M \rightarrow T$
  - Family $V = \{V_k : k \in K\}$ of verification functions V_k: M x T $\rightarrow$ {valid,invalid}
- $V_k(m,t)$ must yield valid iff t is a valid authentication tag for m and k (i.e., $t = A_k(m)$)
- Alternatively speaking, $V_k(m,A_k(m))$ must yield valid ($\forall k \in K$, $m \in M$)
- Typiaclly, $M = \{0,1\}^*$, $T = \{0,1\}^{l_{tag}}$ for some fixed tag length l_{tag}, and $K = \{0,1\}^{l_{key}}$ for some fixed key length l_{key} (e.g., $l_{tag} = l_{key} = 128$)

Message Authentication

Introduction

- Informally speaking, a message authentication system is secure if an adversary has no better possibility to generate a valid MAC than to guess
- The probability of correctly guessing a valid MAC is $1/2^{l_{tag}}$
- More formally speaking, one must define
  - The adversary one has in mind (including, for example, the types of attacks he or she is able to mount)
  - The task he or she must solve in order to be successful, i.e., to break the security of the system

Message Authentication

Introduction

- Types of attacks
  - Known-message attack, i.e., adversary knows (m_i,t_i) for $i = 1,\dots,n$
  - Chosen-message attack, i.e., adversary can choose $m_1,\dots,m_n$
    - Adaptive chosen-message attack
    - Nonadaptive chosen-message attack

  A tag-only attack does not make sense (because the message is always transmitted together with the tag)

- Tasks to solve
  - Total break (retrieve the secret key)
  - Selective forgery
  - Existential forgery

Message Authentication
Introduction

- Strong security definition
  - Even for an adversary who is able to mount an adaptive chosen-message attack, it is *impossible* or *computationally infeasible* to selectively (or existentially) forge a MAC with a success probability that is substantially greater than guessing
    - *Impossible* → the message authentication system is **information-theoretically** or **unconditionally secure**
    - *Computationally infeasible* → the message authentication system is **computationally** or **conditionally secure**

Message Authentication

4.2 Computationally Secure MACs

- Constructions
  - MACs using symmetric encryption systems
  - MACs using keyed hash functions
  - MACs using PRFs
  - MACs based on universal hashing
- There are a few outdated standards, such as the **message authenticator algorithm (MAA)** specified in ISO 8731-2
- The MAA generates MACs that are only 32 bits long (too short to be secure)

Computationally Secure MACs – MACs Using Symmetric Encryption Systems

- If a block cipher is used in CBC (or OFB) mode, then the last ciphertext block (functionally) depends on all previous blocks
- This implies that the last ciphertext block may also serve as MAC (**CBC residue** or **CBC-MAC**)
- The length of a CBC-MAC depends on the block cipher in use (e.g., 64 bits for DES or 3DES, 128 bits for AES, …)
- Relevant standardization bodies and standards in this area
  - ANSI (e.g., ANSI X9.9, ANSI X9.19, …)
  - NIST (e.g., FIPS PUB 113)
  - ISO (e.g., ISO 8730/8731, ISO/IEC 9797, …)
- More recently, the NIST has specified the **cipher-based MAC (CMAC)** as a block cipher mode of operation for authentication (NIST SP 800-38B, IETF Informational RFC 4493, …)

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- The idea of using cryptographic hash functions to protect the authenticity and integrity of data and/or program files dates back to the late 1980s (in the context of virus protection)
- In the early 1990s, people started to think more seriously about the use of such functions to authenticate messages
- Advantages are related to simplicity, efficiency, and exportability
- Initially, there were 3 methods to authenticate a message $m \in \mathcal{M}$ using a cryptographic hash function h (with compression function f) and a secret key $k \in \mathcal{K}$ (or a pair of keys $(k_1,k_2) \in \mathcal{K}$, respectively)
  - Secret prefix method → $MAC_k(m) = h(k \| m)$
  - Secret suffix method → $MAC_k(m) = h(m \| k)$
  - Envelope method → $MAC_{k_1,k_2}(m) = h(k_1 \| m \| k_2)$
- All methods are vulnerable or have structural weaknesses

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- The **secret prefix method**

  $$MAC_k(m) = h(k \| m)$$

  is vulnerable to a message extension or padding attack
- If an adversary knows a message-MAC-pair $(m, h(k \| m))$ with $m = m_1 \| m_2 \| \ldots \| m_i$, then he or she can selectively forge a MAC for a message m‘ with an additional block m_{i+1} of data, i.e., $m' = m \| m_{i+1}$

  $$MAC_k(m') = f(MAC_k(m) \| m_{i+1})$$

- The adversary can therefore forge a valid MAC without having to know either k or how to otherwise compute $MAC_k(m')$
- The attack can be repeated for multiple message blocks

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- There are several possibilities to protect against the message extension or padding attack
  - Only part of the hash value is taken as output (e.g., 64 bits)
  - The messages to be hashed are of fixed length
  - An explicit length field is included at the beginning of the message that is authenticated
- All possibilities have disadvatages in practical use

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- The **secret suffix method**

$$MAC_k(m) = h(m \| k)$$

has a structural weakness

- $MAC_k(m)$ refers to

$$\underbrace{f(f(f(\ldots f(f(m_1) \| m_2) \| \ldots) \| m_i)}_{h^*(m)} \| k)$$

where $h^*(m)$ represents the hashed message m (without initialization and padding)

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- Note that h*(m) does not depend on k, and hence anybody can determine this value for a given m
- This possibility can be turned into a (partly) known-message attack
- Whether this poses a problem depends on the compression function f in use
- The compression functions of the deployed cryptographic hash functions seem to be resistant
- But one need not take the risk

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- The **envelope method**

$$MAC_{k_1,k_2}(m) = h(k_1 \| m \| k_2)$$

  combines the prefix and suffix methods
- For quite a long time, the envelope method was assumed to be secure (in the sense that breaking it requires a simultaneous exhaustive key search for k_1 and k_2)
- In 1995, it was shown that there are more efficient attacks and alternative constructions were proposed

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- Most importantly, the **hashed MAC (HMAC)** construction specified in RFC 2104 is important and widely deployed today

$$HMAC_k(m) = h(k \oplus opad \| h(k \oplus ipad \| m))$$

- ipad and opad are (inner and outer) padding values
  - ipad consists of byte `0x36` (i.e., {`00110110`}) repeated 64 times
  - opad consists of byte `0x5C` (i.e., {`01011100`}) repeated 64 times
- Consequently, ipad and opad comprise $64 \cdot 8 = 512$ bits
- To improve the efficiency of an HMAC implementation, $(k \oplus opad)$ and $(k \oplus ipad)$ can be precomputed
- Also, the output can be truncated to a shorter value (e.g., HMAC-MD5-96 as used for IPsec)

Message Authentication

Computationally Secure MACs – MACs Using Keyed Hash Functions

- Exercise 4-1: Message authentication

  1. Use CrypTool to visualize the generation of MACs (CrypTool > Indiv. Procedures > Hash > Generation of MACs…)
  2. Try to match the CrypTool MAC variants to the method discussed so far

Message Authentication

Computationally Secure MACs – MACs Using PRFs

- MACs using keyed hash functions have the structural weakness that they cannot be computed and verified in parallel (i.e., they must be computed and verified sequentially)
- In 1995, it was proposed to replace iterated hash functions with a family F of PRFs f_k to compute and verify MACs in parallel
- Such a family F can, for example, be constructed with a block cipher (e.g., DES)
- Each encryption function $DES_k(\cdot)$ represents a function f_k from the PRF family F
- The basic idea of the XOR MAC construction is to apply a PRF f_k to each block of a message $m = m_1 \| m_2 \| \ldots \| m_n$ in parallel, and then to add all images $f_k(m_1), f_k(m_2), \ldots, f_k(m_n)$ modulo 2

$$\text{XOR MAC}_k(m) = f_k(m_1) \oplus f_k(m_2) \oplus \ldots \oplus f_k(m_n)$$

Message Authentication

Computationally Secure MACs – MACs Based on Universal Hashing

- Universal hashing provides a general design paradigm for MACs and message authentication systems
- Its main advantages are efficiency and provability
- In universal hashing, one considers classes (or families) of hash functions $h: X \rightarrow Y$
- A class (or family) H of hash functions $h: X \rightarrow Y$ is universal (or two-universal, respectively) if for every $x,y \in X$ with $x \neq y$

$$Pr_{h \in H}[h(x) = h(y)] \leq 1/|Y|$$

- In some sense, the hash functions of H are as collision-resistent as possible (given the cardinality of Y).

Message Authentication

Computationally Secure MACs – MACs Based on Universal Hashing

- The basic idea of a MAC based on universal hashing – aka **universal MAC (UMAC)** – is to
  - Randomly choose a hash function from a two-universal class (or family) H of hash functions
  - Apply this hash function to the message
  - Encrypt the resulting hash value (of the message)
- The encryption algorithm may be the addition modulo 2, where the key stream is derived from a nonce r using a PRF F
- The UMAC construction employs 2 keys k_1 and k_2, as well as a fresh (but not secret) nonce r to authenticate message m

$$UMAC_{k_1,k_2}(m) = h_{k_1}(m) \oplus f_{k_2}(r)$$

Message Authentication

4.3 Information-theoretically Secure MACs

- There are MACs and message authentication systems that are information-theoretically secure
- This means that no matter how much computational power the adversary has, he or she is not able to forge a valid MAC with a success probability substantially better than guessing
- But information-theoretically secure MACs require a new key for every message to be authenticated (similar to information-theoretically secure symmetric encryption systems)
- If one employs a (cryptographically strong) PRBG to generate the keys, then the resulting MACs are only computationally secure (again, this is similar to the one-time pad)
- The bottom line is that information-theoretically secure MACs are theoretically interesting but not very useful in practice

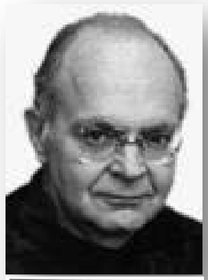
Message Authentication

4.4 Final Remarks

- Computationally secure MACs are widely deployed in practice
- Most applications and standards employ MACs using keyed hash functions
- Most importantly, the HMAC construction is part of most Internet security protocols (e.g., IPsec, SSL/TLS, ...)
- The fact that the HMAC construction is based on iterated (crypto-graphic) hash functions may lead to performance problems – especially in high-speed networks
- MAC constructions that can be parallelized (e.g., XOR MAC) provide alternatives
- MACs based on universal hashing (e.g., UMAC) are promising because they are highly efficient and their security properties can be proven

5 Random and Pseudorandom Bit Generators

- Donald E. Knuth
  - *Random numbers should not be generated with a method chosen at random*
- John von Neumann (1903 – 1957)
  - *Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin*



Random and Pseudorandom Bit Generators

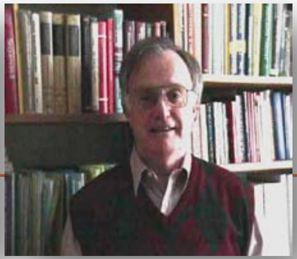
5.1 Introduction

- The term **randomness** refers to nondeterminism
- We say that something is **random**, if its outcome is non-deterministic, meaning that it cannot be predicted in some meaningful way
- Whether randomness really exists is a philosophical question
- According to present knowledge in quantum physics, we believe that randomness exists
- Even if randomness exists, it is not clear how to measure it

Random and Pseudorandom Bit Generators

Introduction

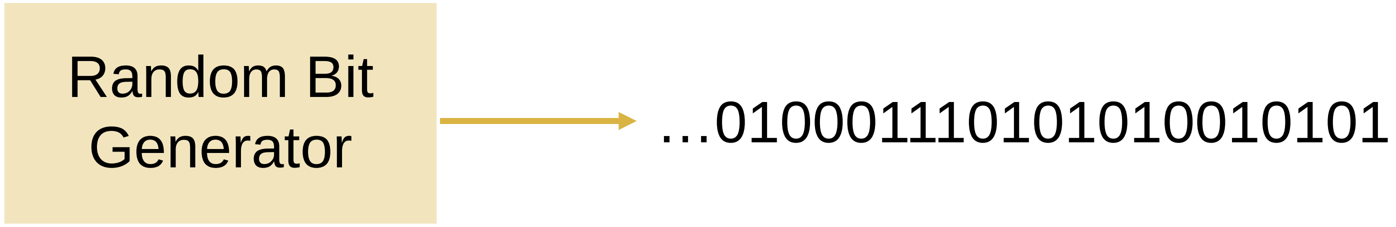
- For a finite sequence of values, the **Kolmogorov complexity** measures its randomness (i.e., it measures the minimal length of a program for a Turing machine that is able to generate the sequence)
- The Kolmogorov complexity is a purely theoretical measure (it is not known how to compute it)
- If a (pseudorandom) bit sequence is generated with a LFSR, then the **linear complexity** measures its randomness (i.e., it measures the size of the shortest LFSR that generates the bit sequence)
- The **Berlekamp-Massey algorithm** can be used to compute the linear complexity (and hence the randomness) of a bit sequence



Random and Pseudorandom Bit Generators

Introduction

- Remember that a **random bit generator** is a(n idealized model of a) device or algorithm that has no input, and that outputs a sequence of statistically independent and unbiased bits → **binary symmetric source (BSS)**



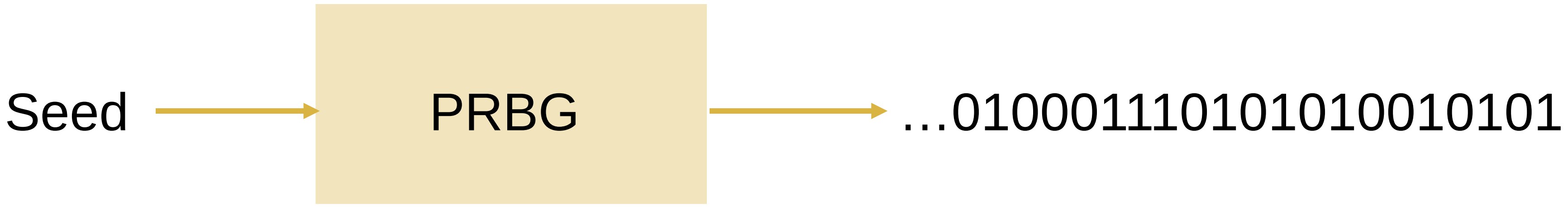
- This means that for $k \geq 1$, all 2^k k-tuples of bits occur with the same probability $1/2^k$
- If one can generate random bits b_i $(i = 1,2,\ldots)$, then one can also construct n-bit random numbers

$$a = \sum_{i=1}^{n} b_i 2^{i-1} \in [0,2^n)$$

Random and Pseudorandom Bit Generators

Introduction

- Also remember that a **PRBG** is an efficient deterministic algorithm that, given as input a truly random binary sequence of length k (seed), generates and outputs a (pseudorandom) bit sequence of length l >> k that appears to be random

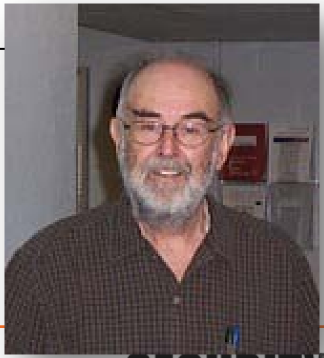


- Hence, the PRBG acts as a „randomness expander“ for the seed (it must still be given a truly random value to start with)
- In practice, most additive stream ciphers (e.g., RC4/ARCFOUR) implement a PRBG, and hence designing an additive stream cipher and designing a PRBG refer to the same problem

Random and Pseudorandom Bit Generators

5.2 Statistical Randomness Testing

- It is not known how to directly test the randomness of a binary (bit) sequence
- It is only known how to indirectly test it by trying to detect certain kinds of defects (using statistical randomness tests)
- If a bit sequence passes a certain set of statistical randomness tests, then the respective generator is accepted as being random (or not rejected as being nonrandom)
- There are several sets of statistical randomness tests
  - Maurice George Kendell and Bernard Babington Smith (1938)
  - George Marsaglia‘s Diehard tests (1995)
  - Pierre L‘Ecuyer and Richard Simard‘s TestU01 (2007)
  - NIST test suite (2010) → comprises Ueli Maurer‘s universal statistical test (1992)



Random and Pseudorandom Bit Generators

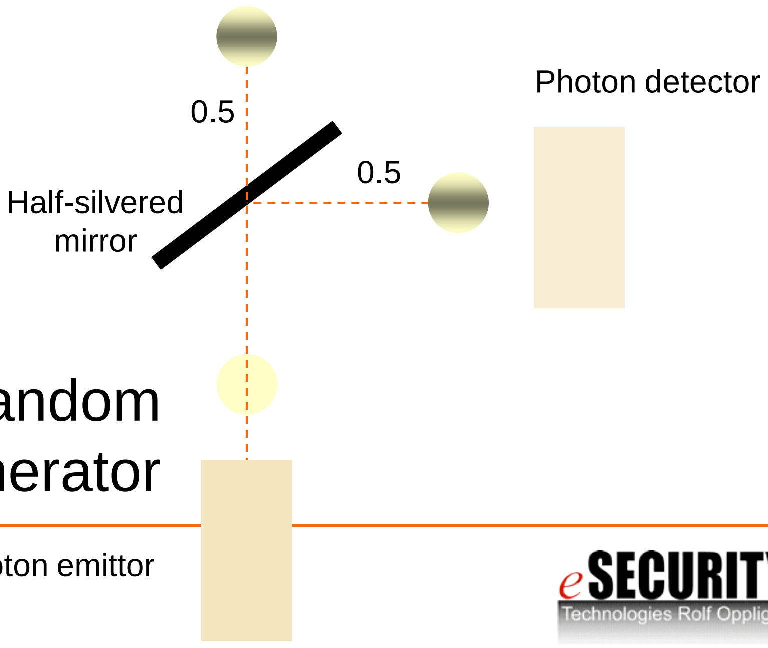
5.3 Random Bit Generators

- The design and implementation of a random bit generator is highly involved
- It is generally recommended to use special hardware to generate truly random bits (e.g., RFC 4086 ≡ BCP 106)
- But there are situations in which special hardware is not available and software must be used to generate random bits
- Hence, there is room for hardware-based and software-based random bit generators

Random and Pseudorandom Bit Generators

Random Bit Generators

- **Hardware-based random bit generators** exploit the randomness that occurs in physical processes and/or phenomena
- Examples
  - Elapsed time between emission of particles during radioactive decay
  - Frequency instability of a free-running oscillator
  - Amount a metal-insulator-semiconductor capacitor is charged during a fixed period of time
  - Air turbulence within a sealed disk drive that causes random fluctuations in disk drive sector read latency time
  - ...



Quantum random bit generator

Random and Pseudorandom Bit Generators

Random Bit Generators

- The design of **software-based random bit generators** is a difficult and challenging (engigeering) task
- There are (internal) states that may be used by a software-based random bit generator
  - System clock (especially in the milliseconds range)
  - Elapsed time between keystrokes and/or mouse movements
  - Contents of I/O buffers
  - Input provided by the user
  - Values of operating system variables
  - ...
- If one does not have one single random source but several uncorrelated sources, then one can use a strong mixing function

Random and Pseudorandom Bit Generators

Random Bit Generators

- Any source of random bits can be defective, i.e., the output bits are biased and/or correlated
- There are **deskewing techniques** for generating truly random bit sequences from the output of a defective random bit generator
- Simple deskrewing technique (John von Neumann)
  - Group the output sequence into pairs of bits
  - Transform `10` to `1` and `01` to `0` (discard `00` and `11`)
- For example, `0110001101110101011010` is transformed to `01--0-00011`

Random and Pseudorandom Bit Generators

Random Bit Generators

- Exercise 5-1: Von Neumann‘s deskewing technique

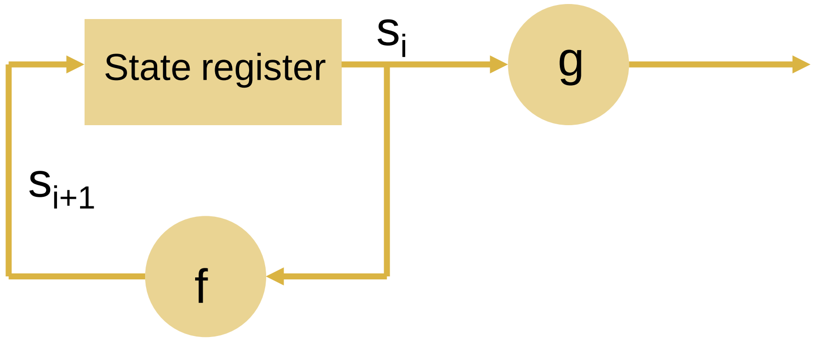
  1. Apply von Neumann‘s deskewing technique to the following output of a (possibly defective) random bit generator

```
1001001001001000100111101011010001010110000101

0110100100011111011010011101010010111101010000

1101001010010100101110101010011010000101110101
```

Random and Pseudorandom Bit Generators

5.4 Pseudorandom Bit Generators

- A PRBG is deterministic and has no other input value than the seed
- A PRBG therefore represents and can be modeled as a **finite state machine (FSM)**
- Components of an FSM
  - State register (initialized with seed s_0)
  - State transition or next-state function f that operates on the state register, i.e., $s_{i+1} = f(s_i)$
  - Output function g that computes an output value x_i (typically a bit or series of bits) from s_i
- The PRBG generates a binary sequence $(x_i)_{i\geq1} = x_1, x_2, x_3, \ldots$ that is cyclic (with a potentially very long cycle)



Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

- If s_i is known, then $(x_j)_{j \geq i}$ is predicatble (this is due to the deter-minsitic nature of the PRBG representing an FSM)
- Hence, some PRBGs used in practice deviate from the idealized model by allowing the state to be reseeded periodically
- This can be modeled by having and taking into account an additional source of randomness
- The bottom line is that the distinction between a true random bit generator and a PRBG is fuzzy in practice

Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

- There are several security requirements for PRBGs
- For example, an obvious (and minimal) security requirement is that the length of s_0 must be so large that an exhaustive search over all $2^{|s_0|}$ possible seed values is computationally infeasible
- Also, the bit sequence generated by the PRBG must pass all relevant statistical randomness tests (necessary but not sufficient)
- But there are PRBGs that pass all relevant statistical randomness tests but are still not sufficiently secure for cryptographic purposes
  - PRBGs that employ the binary expansion of algebraic numbers, such as SQRT(3), SQRT(5), …
  - Linear congruential generators that generate $(x_i)_{i \geq 1}$ using the linear recurrence $x_i \equiv ax_{i-1}+b \pmod{n}$ for seed $x_0 = s_0$ and integers a, b, and n

Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

- In general, PRBGs that employ a LFSR are not sufficiently secure
- But there are **LFSR-based PRBGs** that are sufficiently secure and that can be used for cryptographic purposes and applications
  - **Shrinking generator**

S	(s_i) =	0	1	1	0	1	1	0	0	1	1	0	1
A	(a_i) =	1	0	1	1	0	1	1	1	0	1	0	0
	Output =		0	1		0	1			0	1		0

In each cycle i, the shrinking generator out-puts a_i iff $s_i = 1$ (otherwise a_i is discarded)

  - **Self-shrinking generator**

A	(a_i) =	$\underline{1}$	0	$\underline{1}$	1	$\underline{0}$	1	$\underline{1}$	1	$\underline{0}$	1	$\underline{0}$	0
	Output =		0		1				1				

In each cycle i, the self-shrinking generator outputs a_{2i+1} iff $a_{2i} = 1$ (otherwise a_{2i+1} is discarded)

Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

- There are alternative PRBGs (not based on LFSRs)
- For example, the **ANSI X9.17 PRBG** is frequently used in practice

```
(s,k,n)
--------------------
$I \leftarrow E_k(D)$
For i = 1 to n do
    $x_i \leftarrow E_k(I \oplus s)$
    $s \leftarrow E_k(I \oplus x_i)$
    output $x_i$
--------------------
$(x_1,x_2,\ldots,x_n)$
```

- E_k denotes 3DES encryption with 2 keys
- D is an internally used 64-bit representation of date/time
- I is an intermediate value
- The output is a sequence of n pseudo-random 64-bit strings

Random and Pseudorandom Bit Generators

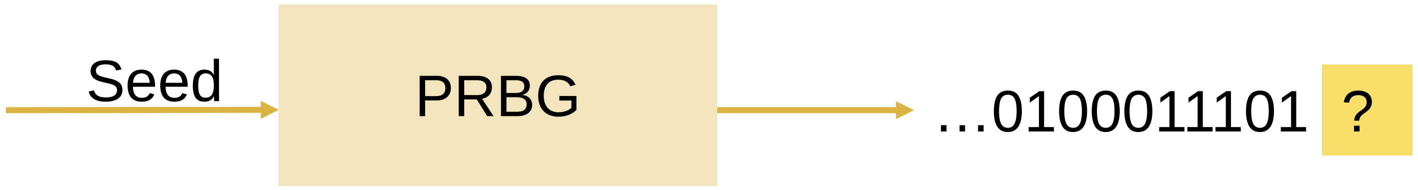
Pseudorandom Bit Generators

- Little is known about the cryptographical strength of the PRBGs used in practice
- For example, many PRBGs (including the ANSI X9.17 PRBG) have the property that once the internal state is compromised, an adversary can forever after predict the output sequence
  - From a theoretical viewpoint, this is a minor concern that does not disturb the security of the PRBG (because one assumes that the adversary is not able to read out internal state in the first place)
  - From a prcatical viewpoint, this is a major concern that requires to reseed the PRBG periodically
- PRBGs that are resistant against known attacks are sometimes called practically strong
- They are usually very efficient (even when implemented in software)

Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

- The notion of a cryptographically secure PRBG was introduced and formalized by Manuel Blum (ACM Turing Award 1995), Silvio Micali, and Andrew Yao (ACM Turing Award 2000) in the early 1980s
- A PRBG is **cryptographically secure** if it passes the **next-bit test**, i.e., an adversary cannot predict the next bit in a bit sequence with a probability substantially better than guessing

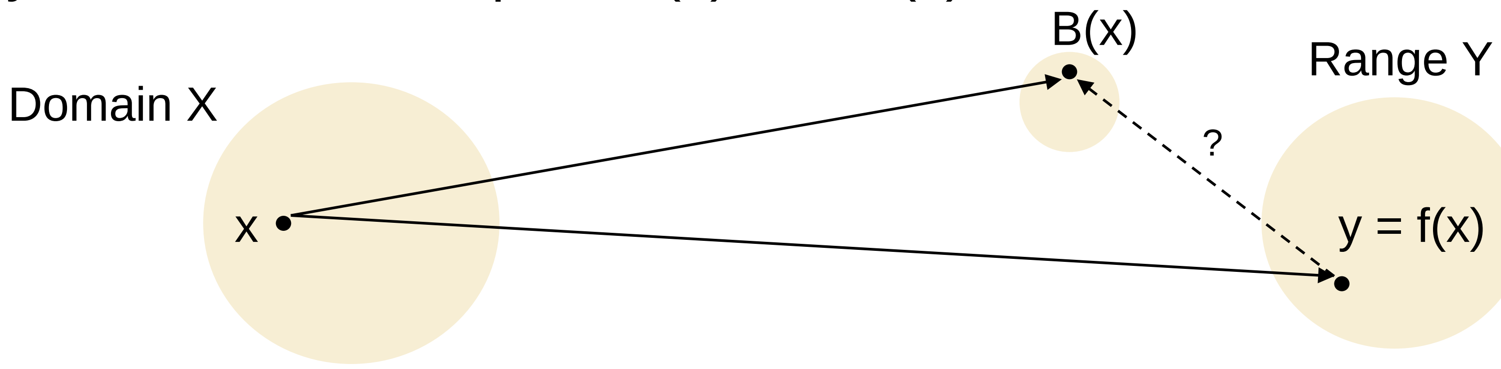


- A PRBG that passes the next-bit test is **perfect** in the sense that no probabilistic polynomial-time (PPT) algorithm can guess with a probability significantly greater than ½ whether an input k-bit string was randomly selected from $\{0,1\}^k$ or pseudorandomly generated with the PRBG

Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

- To construct a cryptographically secure PRBG, one needs a one-way function $f: X \rightarrow Y$ with hard-core predicate B, i.e., it is computationally infeasible to compute B(x) from f(x)



- Starting with a seed s_0, a cryptographically secure PRBG G (with stretching function l) can be constructed as

$$G(s_0) = B(f(s_0)),B(f^2(s_0)),\ldots,B(f^{l(|s_0|)}(s_0))$$

- There are many instantiations of this construction idea (using different one-way functions and respective hard-core predicates)

Random and Pseudorandom Bit Generators

Pseudorandom Bit Generators

Blum-Micali PRBG

(p,g)

```
$x_0 \in_R \mathbf{Z}_p^*$
for i = 1 to ∞ do
    $x_i \leftarrow g^{x_{i-1}} \pmod{p}$
    $b_i \leftarrow msb(x_i)$
    output $b_i$
```

$(b_i)_{i \geq 1}$

RSA PRBG

(n,e)

```
$x_0 \in_R \mathbf{Z}_n^*$
for i = 1 to ∞ do
    $x_i \leftarrow x_{i-1}^e \pmod{n}$
    $b_i \leftarrow lsb(x_i)$
    output $b_i$
```

$(b_i)_{i \geq 1}$

Blum-Blum-Shub (BBS) PRBG

(n)

```
$x_0 \in_R \mathbf{Z}_n^*$
for i = 1 to ∞ do
    $x_i \leftarrow x_{i-1}^2 \pmod{n}$
    $b_i \leftarrow lsb(x_i)$
    output $b_i$
```

$(b_i)_{i \geq 1}$

Random and Pseudorandom Bit Generators

5.5 Final Remarks

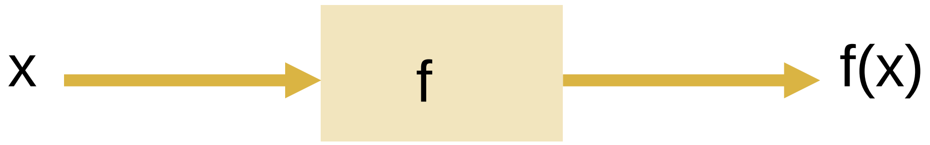
- (Hardware- or software-based) random bit generators are at the core of many cryptographic systems and applications
  - Deskewing techniques may be used to correct any defectiveness
  - Statistical randomness testing may be used to evaluate the quality of the output
- In practice, it is often required that a random bit generator conforms to a security level specified in FIPS PUB 140-2
- From an application viewpoint, random bit generators are used to seed PRBGs, and PRBGs are then used to generate (potentially infinite) sequences of pseudorandom bits
- The underlying assumption is that a cryptographic system that is secure when a true random bit generator is used remains secure when „only“ a (reasonably strong or secure) PRBG is used

6 Random and Pseudorandom Functions

Random and Pseudorandom Functions

6.1 Introduction

- Before one can meaningfully introduce and discuss the notion pseudorandom functions, one must introduce the notion of a random function
- In short, a **random function** (aka **random oracle**) f is an arbitrary mapping from domain X to range Y, i.e., f: X $\rightarrow$ Y



- This suggests that f maps an input value x$\in$X to an arbitrary output value f(x)$\in$Y
- The only requirement is that the same input value x is always mapped to the same output value f(x)

Random and Pseudorandom Functions

Introduction

- Another way to think about a random function f is a large random table T with randomly assigned entries T[x] = (x,f(x))

x	f(x)
x_1	$f(x_1)$
x_2	$f(x_2)$
...	...

- Note that the term „random function“ is somehow misleading, and that the attribute „random“ does not refer to the output of the function but rather to the way it is chosen
- Also note that a random function is only a conceptual and theoretical construct (i.e., it is not intended to be implemented in practice)

Random and Pseudorandom Functions

Introduction

- A **pseudorandom function (PRF)** is a function that behaves like a random function
- This means that there is a family $F = \{f_s \mid s \in \mathbf{N}\}$ of PRFs, and that each seed value $s \in \mathbf{N}$ selects and determines a particular PRF from F
- Without knowing s, f_s looks like a function that is randomly chosen from F

Random and Pseudorandom Functions

6.2 Constructions

- PRF families and PRBGs are related in the sense that a PRF family can be used to construct a PRBG and a PRBG can be used to construct a PRF family
- Given a PRF family $F = \{f_s | s \in \mathbf{N}\}$, a **PRF-based PRBG** G can, for example, be constructed as $G(s) = (f_s(i))_{i \geq 0} = f_s(0), f_s(1), f_s(2)$
- Given a PRBG G(s), the construction of a **PRBG-based PRF family** is not obvious
- A respective construction is due to Oded Goldreich, Shafi Goldwasser, and Silvio Micali

Random and Pseudorandom Functions

Constructions

- Preliminaries
  - $X = Y = \{0,1\}^n$
  - $x = \sigma_n \cdots \sigma_2 \sigma_1$ is the bitwise representation of x
  - G(s) is a PRBG with stretching function $l(n) = 2n$
  - $G_0(s)$ refers to the first n bits of G(s) for $s \in \{0,1\}^n$
  - $G_1(s)$ refers to the last n bits of G(s) for $s \in \{0,1\}^n$
- A PRBG-based PRF $f_s(x)$ can be defined as $f_s(x) = f_s(\sigma_n \cdots \sigma_2 \sigma_1) = G_{\sigma_n}(\cdots G_{\sigma_2}(G_{\sigma_1}(s))\cdots)$

- Example
  - $n = 2$
  - PRBG G(s)
    - o G(00) = 1001
    - o G(01) = 0011
    - o G(10) = 1110
    - o G(11) = 0100
  - For s = 10 and x = 01, $f_s(x) = f_{10}(01) = G_0(G_1(10)) = G_0(10) = 11$
  - Note that $G_1(10) = G_1(11\underline{10}) = 10$ and $G_0(10) = G_0(\underline{11}10) = 11$

Random and Pseudorandom Functions

6.3 Random Oracle Model

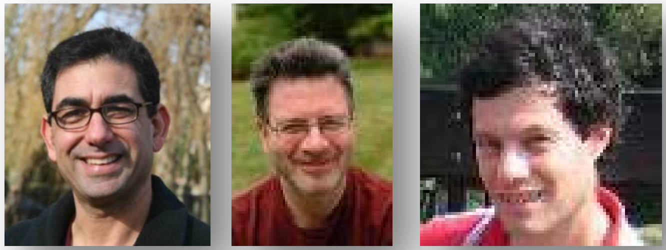
- The **random oracle methodology** was proposed by Mihir Bellare and Phil Rogaway in the 1990s to prove the security properties of cryptographic systems (mainly cryptographic protocols)

- Methodology
  1. A publicly known random function is used to design an ideal system
  2. The security of the ideal system is formally shown
  3. The random function is replaced with a publicly known family of PRFs (typically a cryptographic hash function)
- The result is a real-world implementation of the ideal system
- It is hoped that the security of the ideal system is maintained in the real world

Random and Pseudorandom Functions

Random Oracle Model

- A formal analysis in the random oracle model is not a security proof (because of the ideality assumption)
- But the analysis provides some evidence for the security of the cryptographic system in question
- In 1998, Ran Canetti, Oded Goldreich and Shai Halevi presented an artificially crafted DSS that is yet secure in the random oracle model, but that gets totally insecure when the random oracle is implemented by any (family of) cryptographic hash function(s)



- Since then, researchers think contraversially about the usefulness of the random oracle methodology and model

Random and Pseudorandom Functions

6.4 Final Remarks

- Random functions and (families of) PRFs are theoretical constructs
- (Families of) PRFs can, for example, be associated with symmetric encryption systems (e.g., DES represents a family of PRFs - each key selects a particular encryption function from the family)
- Families of PRFs are conceptually related to PRBGs (if you can contruct one, you can also contruct the other)
- The random oracle methodolgy can be used to design and argue about the security of cryptographic systems (typically cryptographic protocols)
- Due to its limitations and shortcomings, researchers nowadays try to avoid the random oracle model when they prove (or analyze) the security of a cryptographic system

Module 3

7 One-Way Functions

One-Way Functions

7.1 Introduction

- Remember that a function $f: X \rightarrow Y$ is **one-way** if $f(x)$ can be computed efficiently for $x \in_R X$, but $f^{-1}(y)$ cannot be computed efficiently for $y \in_R Y$
- The function can be computed efficiently, if a respective **probabilistic polynomial-time (PPT) algorithm** is known
- More formally, a function $f: X \rightarrow Y$ is one-way if the following two conditions are fulfilled
  - – f is easy to compute, i.e., $f(x)$ can be computed efficiently $\forall x \in X \Rightarrow \exists$ PPT algorithm A that outputs $A(x) = f(x)$ $\forall x \in X$
  - – f is hard to invert, i.e., it is not known how to efficiently compute $f^{-1}(f(x))$ $\forall x \in X$ or $f^{-1}(y)$ for $y \in_R Y \Rightarrow \neg \exists$ PPT algorithm A that outputs $A(f(x)) = f^{-1}(f(x))$ $\forall x \in X$ or $A(y) = f^{-1}(y)$ for $y \in_R Y$

In either case, A is not required to find the correct x – it must only find some inverse of y = f(x)

If f is injective, then x is the only inverse of y = f(x)

One-Way Functions

Introduction

- Alternative ways of speaking (or formalizations)
  - Any PPT algorithm A attempting to invert f on an element of its range can only succeed with a probability that is negligible, i.e.,

    $$\Pr[A(f(x),1^n)] \in f^{-1}(f(x))] \leq 1/p(n)$$

    for every PPT algorithm A, $x \in X$, polynomial p, and sufficiently large n (representing the binary length of x)
  - If x is selected uniformly from $\{0,1\}^n$, y is assigned f(x), and z is assigned $A(y,1^n)$, then the probability that f(z) equals y is negligible

    $$\Pr[f(z) = y : x \leftarrow \{0,1\}^n;\ y \leftarrow f(x);\ z \leftarrow A(y,1^n)] \leq 1/p(n)$$

One-Way Functions

Introduction

- A one-way function f: $X \rightarrow X$ (i.e., Y=X) is a **one-way permutation**
- A one-way function f: $X \rightarrow Y$ is a **trapdoor function** or **trapdoor one-way function**, if there is a trapdoor information t and a PPT algorithm I that can be used to efficiently compute x' = I(f(x),t) with f(x') = f(x)

One-Way Functions

Introduction

- Many cryptographic functions required to be one way map input bitstrings of maximum length to output bitstrings of fixed size (e.g., cryptographic hash function)
- Given such a function, one may ask how expensive it is to invert it
- If the maximum allowable length is n, then 2^n tries are probably sufficient to find a preimage for a given value
- Because 2^n is constant for $n \in \mathbf{N}$, the computational complexity to invert the function is still O(1)
- Hence, if one wants to use complexity-theoretic arguments, then one cannot work with a fixed (and constant) value of n
- Instead, it must be possible to let n grow arbitrarily large
- Hence, one must consider families of (one-way) functions and there must be at least one function for every $n \in \mathbf{N}$

One-Way Functions

Introduction

- A family of functions $F = \{f_i: X_i \rightarrow Y_i\}_{i \in I}$ represents a **family of one-way functions** if the following three conditions are fulfilled
  - I is an infinite index set
  - Every $i \in I$ selects a function $f_i: X_i \rightarrow Y_i$ from the family F
  - Every function f_i is one way
- A family of one-way functions $F = \{f_i: X_i \rightarrow Y_i\}_{i \in I}$ is a **family of one-way permutations** if every f_i is a one-way permutation (i.e., $X_i = Y_i$)
- A family of one-way functions $F = \{f_i: X_i \rightarrow Y_i\}_{i \in I}$ is a **family of trapdoor (one-way) functions** if every f_i is a trapdoor (one-way) function with trapdoor information t_i
- A family of one-way functions $F = \{f_i: X_i \rightarrow Y_i\}_{i \in I}$ is a **family of trapdoor (one-way) permutations** if every f_i is a trapdoor (one-way) permutation over X_i (i.e., $X_i = Y_i$) with trapdoor information t_i

One-Way Functions

7.2 Candidate One-Way Functions

- Pro memoria: Functions that are assumed (or conjectured) to be one-way
  - Discrete exponentiation (Exp): $y = f(x) = g^x \pmod{p}$
  - Modular power (RSA): $y = f(x) = x^e \pmod{n}$
  - Modular square (Square): $y = f(x) = x^2 \pmod{n}$
- Remember that none of these functions has been proven to be one-way
- Also remember that it is theoretically not even known whether one-way functions exist (i.e., a proof of existence is still missing)

One-Way Functions

Candidate One-Way Functions – Exp

- If p is a prime and g is a generator (primitive root) of $\mathbf{Z}_p^*$, then

$$\begin{aligned} \mathrm{Exp}_{p,g}: \mathbf{Z}_{p-1} &\rightarrow \mathbf{Z}_p^* \\ x &\rightarrow g^x \pmod p \end{aligned}$$

  is the **discrete exponentiation function** to the base g

- $\mathrm{Exp}_{p,g}$ defines a group isomorphism from $<\mathbf{Z}_{p-1},+>$ to $<\mathbf{Z}_p^*,\cdot>$

$$\mathrm{Exp}_{p,g}(x+y) = \mathrm{Exp}_{p,g}(x)\cdot\mathrm{Exp}_{p,g}(y)$$

$$g^{x+y} \equiv g^x \cdot g^y \pmod p$$

- Because $\mathrm{Exp}_{p,g}$ is a bijective function, it must have an inverse function

One-Way Functions

Candidate One-Way Functions – Exp

- The inverse function

$$\begin{aligned} \mathrm{Log}_{p,g}: \mathbf{Z}_p^* &\rightarrow \mathbf{Z}_{p-1} \\ x &\rightarrow \log_g x \end{aligned}$$

  is the **discrete logarithm function** to the base g
- The value $\log_g x$ refers to the discrete logarithm of x to the base g
- Contrary to $\mathrm{Exp}_{p,g}$, no efficient algorithm is known to compute $\mathrm{Log}_{p,g}$ for sufficiently large p (i.e., the best known algorithms have super-polynomial running time)
- To use complexity-theoretic arguments, one must consider a family of one-way functions

One-Way Functions

Candidate One-Way Functions – Exp

- All possible pairs (p,g) define an index set I

$$I := \{(p,g) \mid p \text{ is prime and } g \text{ is a generator of } \mathbf{Z}_p^*\}$$

- A family of discrete exponentiation functions can be defined as follows:

$$Exp := \{Exp_{p,g}: \mathbf{Z}_{p-1} \rightarrow \mathbf{Z}_p^*, x \rightarrow g^x \pmod{p}\}_{(p,g)\in I}$$

- A respective family of discrete logarithms functions can be defined as follows

$$Log := \{Log_{p,g}: \mathbf{Z}_p^* \rightarrow \mathbf{Z}_{p-1}, x \rightarrow \log_g x\}_{(p,g)\in I}$$

- Exp represents a family of one-way functions (with no known trapdoor)

One-Way Functions

Candidate One-Way Functions – Exp

- The one-way property of Exp is based on the **discrete logarithm assumption (DLA)**
- The DLA suggests that there is no known algorithm to efficiently compute $\text{Log}_{p,g}$ for appropriately chosen p and g
- Problems related to the DLA
  - Discrete logarithm problem (DLP)
  - (Computational) Diffie-Hellman problem (DHP)
  - Decisional Diffie-Hellman problem (DDHP)
- To specify the problems, it is appropriate to abstractly consider a cyclic group G with generator g (i.e., ‹g› = G)
- To give examples, it is more appropriate to employ a specific group, such as $\mathbf{Z}_7^* = \{1,2,3,4,5,6\}$, and a specific generator, such as g = 5

One-Way Functions

Candidate One-Way Functions – Exp

- Let G be a cyclic group, g a generator of G, and h an arbitrary element of G
- The **discrete logarithm problem (DLP)** is to determine an $x \in \mathbf{N}$ such that $g^x = h$
  - Input: p, g, h
  - Output: x such that $g^x \pmod{p} = h$
- Example
  - Solve the DLP for p = 7, g = 5, and h = 4, i.e., find an $x \in \mathbf{N}$ such that $5^x \pmod{7} \equiv 4$
  - The solution is 2 (note that $5^2 \pmod{7} \equiv 25 \pmod{7} \equiv 4$)

One-Way Functions

Candidate One-Way Functions – Exp

- Let G be a cyclic group, g a generator of G, and x and y arbitrary elements of G
- The **(computational) Diffie-Hellman problem (DHP)** is to determine g^{xy} from g^x and g^y
  - Input: p, g, g^x, g^y
  - Output: g^{xy}
- Example
  - Solve the DHP for p = 7, g = 5, $g^x = 6$ (x = 3), and $g^y = 2$ (y = 4)
  - The solution is 1 (note that $5^{3 \cdot 4} \pmod 7 \equiv 5^{12} \pmod 7 \equiv 244{,}140{,}625 \pmod 7 \equiv 4$)
- As its name suggests, the DHP is at the core of the Diffie-Hellman key exchange protocol

One-Way Functions

Candidate One-Way Functions – Exp

- Let G be a cyclic group, g a generator of G, and x, y, and z arbitrary elements of G
- The **decisional Diffie-Hellman problem (DDHP)** is to determine whether g^{xy} or g^z solves the DHP for g^x and g^y
- Alternatively speaking, the DDHP is to distinguish $\langle g^x, g^y, g^{xy} \rangle$ and $\langle g^x, g^y, g^z \rangle$
  - Input: p, g, g^x, g^y, g^{xy}, g^z
  - Output: g^{xy} or g^z

Is g^{xy} or g^z a solution for the DHP with input parameters p, g, g^x, and g^y?

- Example
  - Solve the DDHP for $p = 7$, $g = 5$, $g^x = 6$ ($x = 3$), $g^y = 2$ ($y = 4$), $g^{xy} = 1$ (see above), and $g^z = 3$ ($z = 5$)
  - The solution is $g^{xy} = 1$

One-Way Functions

Candidate One-Way Functions – Exp

- If one can solve the DLP, then one can also solve the DHP and the DDHP (i.e., the DLP is the hardest DLA-based problem)
- More generally, it is known that

$$\text{DDHP} \leq_P \text{DHP} \leq_P \text{DLP}$$

- The exact relationships depend on the cyclic group in use

In many groups, the DHP and DLP are known to be computationally equivalent, i.e., DHP $\equiv_P$ DLP

In some groups, the DHP is known to be more difficult to solve than the DDHP, i.e., the DDHP can be solved in polynomial time, whereas the DHP still requires subexponential time

One-Way Functions

Candidate One-Way Functions – RSA

- If n is the product of 2 distinct primes (i.e., n = pq) and e is relatively prime to $\phi(n)$, then

$$\begin{aligned} RSA_{n,e}: \mathbf{Z}_n^* &\rightarrow \mathbf{Z}_n^* \\ x &\rightarrow x^e \pmod{n} \end{aligned}$$

  is the **RSA function** (permutation over $\mathbf{Z}_n^*$)

- If d is the multiplicative inverse of e modulo $\phi(n)$, then

$$\begin{aligned} RSA_{n,d}: \mathbf{Z}_n^* &\rightarrow \mathbf{Z}_n^* \\ x &\rightarrow x^d \pmod{n} \end{aligned}$$

  is the inverse RSA function of $RSA_{n,e}$

One-Way Functions

Candidate One-Way Functions – RSA

- Again, to use complexity-theoretic arguments, one must consider families of such functions
- The index set I is defined as follows:

$$I := \{(n,e) \mid n = pq;\ p \text{ and } q \text{ prime};\ p \neq q;\ 0<e<\phi(n);\ (e,\phi(n))=1\}$$

- The RSA family comprises all RSA functions referring to I:

$$RSA := \{RSA_{n,e}: \mathbf{Z}_n^* \rightarrow \mathbf{Z}_n^*, x \rightarrow x^e \pmod n\}_{(n,e)\in I}$$

- Since $RSA_{n,e}$ is a permutation over $\mathbf{Z}_n^*$, RSA represents a family of trapdoor (one-way) permutations
- There are multiple trapdoors
  - d
  - (p,q)
  - $\phi(n)$

One-Way Functions

Candidate One-Way Functions – RSA

- The one-way property of RSA is based on the RSA assumption
- The **RSA assumption** suggests that the probability that the RSA function can be inverted without knowing a trapdoor is negligible
- This also suggests that the RSA problem is intractable
- The **RSA problem (RSAP)** is to determine m from n, e, and $c \equiv m^e \pmod{n}$
- Hence, the RSAP is to compute the eth root of c modulo n (without knowing a trapdoor)
- Because the prime factors of n (i.e., p and q) represent a trapdoor for $RSA_{n,e}$, somebody who is able to factorize n can trivially compute $RSA_{n,d}$ and invert $RSA_{n,e}$ accordingly
- One must make the **integer factoring assumption (IFA)**, i.e., for a sufficiently large n it is computationally infeasible to factorize it

One-Way Functions

Candidate One-Way Functions – RSA

- The IFA suggests that the **integer factoring problem (IFP)** is intractable
- The IFP is to determine the prime factors of n, i.e., determine $p_1,\ldots,p_k$ and $e_1,\ldots,e_k \in \mathbf{N}$ such that $n = p_1^{e_1} \ldots p_k^{e_k}$
- It is known that RSAP $\leq_P$ IFP
- Hence, anybody who can solve the IFP can solve the RSAP (and invert the RSA function accordingly)
- The converse is not known to be true, i.e., it is not known whether there exists a simpler way to invert the RSA function than to solve the IFP
- This means that somebody may break RSA without necessarily be able to factorize the modulus n (this may be worrisome)

One-Way Functions

Candidate One-Way Functions – Square

- The modular square function (Square) looks similar to the RSA function (with e = 2)
- Modular square roots can be computed efficiently iff the prime factorization of n is known
- This means that computing square roots in $\mathbf{Z}_n^*$ and factoring n are computationally equivalent (this is in contrast to computing e^th roots in $\mathbf{Z}_n^*$)
- The modular square function is bijective if the domain and range are restricted to QR_n, where n is a Blum integer, i.e., n = pq with p and q distinct primes and $p \equiv q \equiv 3 \pmod 4$

One-Way Functions

Candidate One-Way Functions – Square

- Hence,

$$\text{Square}_n: QR_n \rightarrow QR_n$$
$$x \rightarrow x^2 \pmod{n}$$

is the **square function**, and

$$\text{Sqrt}_n: QR_n \rightarrow QR_n$$
$$x \rightarrow x^{1/2} \pmod{n}$$

is the **square root function**

- Both Square_n and Sqrt_n represent permutations over QR_n
- Once again, to use complexity-theoretic arguments, one must consider families of such functions

One-Way Functions

Candidate One-Way Functions – Square

- The index set I can be defined as follows:

$$I := \{n \mid n = pq; \text{ p and q prime}; p \neq q; |p| = |q|; p,q \equiv 3 \pmod{4}\}$$

- The Square family comprises all Square functions referring to I:

$$\text{Square} := \{\text{Square}_n: QR_n \rightarrow QR_n, x \rightarrow x^2 \pmod{n}\}_{(n,e) \in I}$$

- Similarly, the Sqrt family comprises all Sqrt functions referring to I:

$$\text{Sqrt} := \{\text{Sqrt}_n: QR_n \rightarrow QR_n, x \rightarrow x^{1/2} \pmod{n}\}_{(n,e) \in I}$$

- The Square family is a family of trapdoor (one-way) permuatations that is used by some public key cryptosystems (e.g., Rabin)
- The respective cryptosystems, however, have some severe drawbacks in practical use

One-Way Functions

7.3 Integer Factorization Algorithms

- Integer factorization algorithms are to solve the IFP
- The IFA suggests that there is no efficient integer factorization algorithm
- Integer factorization algorithms
  - **Special-purpose algorithms** depend upon and take advantage of special properties of n
    - Trial division
    - P-1 algorithm (John Pollard, 1974)
      - P+1 algorithm (Hugh Williams, 1982)
      - Elliptic curve method (Hendrik Lenstra, 1987)
    - Pollard Rho (John Pollard, 1975)

One-Way Functions

Integer Factorization Algorithms

- **General-purpose algorithms** depend on nothing and work equally well for all n
    - Continued fraction
    - Quadratic sieve (QS, John Dixon, 1981, and Carl Pomerance, 1984)
    - Number field sieve (NFS, Hendrik and Arjen Lenstra, 1993)
        - Special number field sieve (SNFS)
        - General number field sieve (GNFS)

- Many general-purpose algorithms consist of two steps, of which one can be parallelized and optimized by specific hardware
    - The Weizmann Institute Key-Locating Engine (TWINKLE)
    - The Weizmann Institute Relation Locator (TWIRL)
    - SHARK
    - Yet Another Sieving Device (YASD)

One-Way Functions

Integer Factorization Algorithms

- The RSA public key cryptosystem was published in the August 1977 issue of *Scientific American*
- As a challenge, USD 100 were offered to anyone who could decrypt a message that was encrypted using a 129-digit modulus (RSA-129)
- RSA-129 was factored in 1994 with a distributed QS algorithm

114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541
=
3490529510847650949147849619903898133417764638493387843990820577
×
32769132993266709549961988190834461413177642967992942539798288533

One-Way Functions

Integer Factorization Algorithms

- Until 2007, RSA Laboratories sponsored the **RSA Factoring Challenge** to learn more about the state of the art in integer factorization
  - In 1999, RSA-512 (155 digits) was factored
  - In 2003, RSA-576 (174 digits) was factored
  - In 2004, a 663-bit number (200 digits) was factored
  - In 2005, RSA-640 (193 digits) was factored
  - In 2010, RSA-768 was factored
  - ...
- The next numbers to factor (in the RSA Factoring Challenge) would have been 896, 1024, 1536, and 2048 bits long

One-Way Functions

7.4 Algorithms for Computing Discrete Logarithms

- The DLA suggests that there is no efficient algorithm for computing discrete logarithms
- Algorithms for computing discrete logarithms
  - **Generic algorithms** work in any group
    - Brute-force search
    - Baby-step giant-step algorithm (Daniel Shanks, 1974)
    - Pollard Rho (John Pollard)
  - **Nongeneric (special-purpose) algorithms** attempt to exploit special properties of the group (e.g., $\mathbf{Z}_p^*$)
    - Index-calculus algorithm
    - NFS

It has been shown by Victor Shoup that $O(|G|^{1/2})$ is a lower bound for the expected running time of any generic algorithm that computes discrete logarithms in a cyclic group, and that improvements are possible only if the factorization of |G| is known

One-Way Functions

Algorithms for Computing Discrete Logarithms

- Due to the existence of the NFS, the state of the art in computing discrete logarithms in $\mathbf{Z}_p^*$ is comparable to the state of the art in factoring integers
- One must work with prime numbers of at least 1,024 bits
- Also, care must be taken that p-1 does not have only small prime factors (otherwise the Pohlig-Hellman algorithm can be invoked)
- If one is not working in $\mathbf{Z}_p^*$, then the nongeneric (special-purpose) algorithms do not work, and the state of the art in computing dis-crete logarithms is worse than the state of the art in factoring integers
- In this case, one must invoke generic algorithms
- This fact is, for example, exploited by **elliptic curve cryptography (ECC)**

One-Way Functions

7.5 Final Remarks

- In spite of their importance for public key cryptography, there are only a few functions – Exp, RSA, and Square – conjectured to be one way
- To use complexity-theoretic arguments, one must consider families of such functions
- RSA and Square have known trapdoors, whereas Exp is not known to have any trapdoor
- Surprisingly, factoring integer n and computing discrete logarithms in $\mathbf{Z}_p^*$ have the same difficulty (or computational complexity, res-pectively)
- In practice, it is sometimes recommended to combine cryptosys-tems that employ different candidate one-way functions (to make them more resilient)

8 Asymmetric Encryption Systems

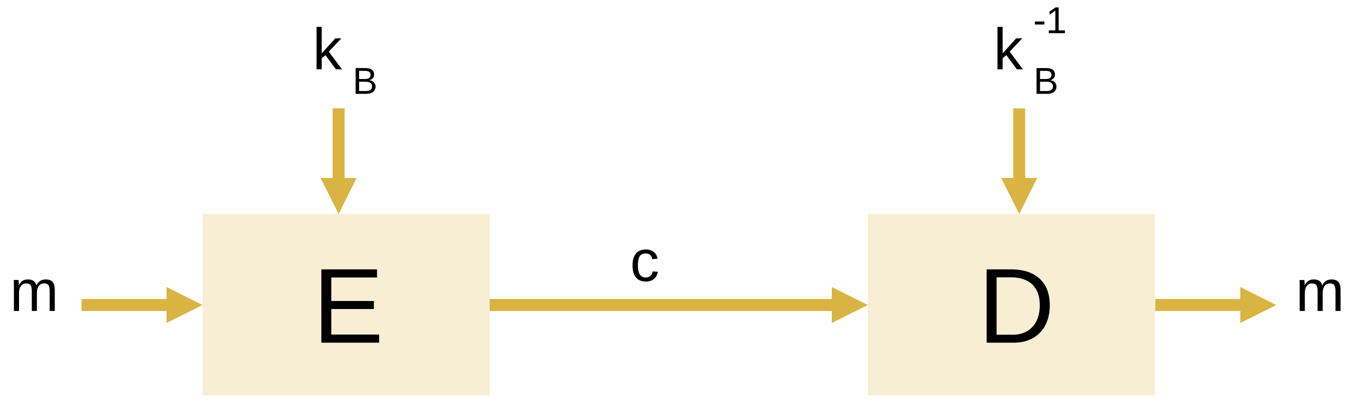
Asymmetric Encryption Systems

8.1 Introduction

- Remember that an **asymmetric encryption system** consists of 3 efficiently computable functions or algorithms
  - **Generate(1^l)** is a probabilistic key generation algorithm that generates a public key pair (k,k^{-1}) based on the security parameter l ($\cong$ key length)
  - **Encrypt(k,m)** is a deterministic or probabilistic encryption algorithm that generates a ciphertext c, i.e., c = Encrypt(k,m)
  - **Decrypt(k^{-1},c)** is a deterministic decryption algorithm that is inverse to Encrypt(k,m) and generates the original plaintext message, i.e., m = Decrypt(k^{-1},c)
- For every public key pair (k,k^{-1}) and plaintext message m, **Encrypt(k,m)** and **Decrypt (k^{-1},c)** must be inverse to each other, i.e., Decrypt (k^{-1},Encrypt(k,m)) = m

Asymmetric Encryption Systems

Introduction



- If k_B and k_B^{-1} do not correspond to each other, then c must decrypt to gibberish
- Long messages must be split into a sequence of message blocks and each block must be encrypted and decrypted individually
- Again, there are modes of operation that can used

Asymmetric Encryption Systems

Introduction

- When discussing the security of an asymmetric encryption system, the first observation is that the notion of information-theoretic or perfect security (according to Shannon) does not make sense
- This is because the Encrypt algorithm employs a public key k, i.e., an adversary who knows c and k can always mount an exhaustive search to find m
- Consequently, the best one can achieve is (some possibly strong form of) conditional or computational security
- To elaborate on the (computational) security of an asymmetric encryption system, one must specify the adversary‘s capabilities and the task required to solve

Asymmetric Encryption Systems

Introduction

– With regard to the adversary‘s capabilities, one typically ssumes a polynomially bounded adversary who can mount specific attacks (with increasing sophistication and power)

  - Ciphertext-only attack
  - Known-plaintext attack
  - (Adaptive) chosen-plaintext attack (CPA)
  - Chosen-ciphertext attacks (CCA)
  - Adaptive chosen-ciphertext attack (CCA2)

Always possible

Asymmetric Encryption Systems

Introduction

- With regard to the task required to solve, there are many possibilities (in addition to simply decrypting a ciphertext)
- The possibilities lead to different notions of security
  - **Semantic security**

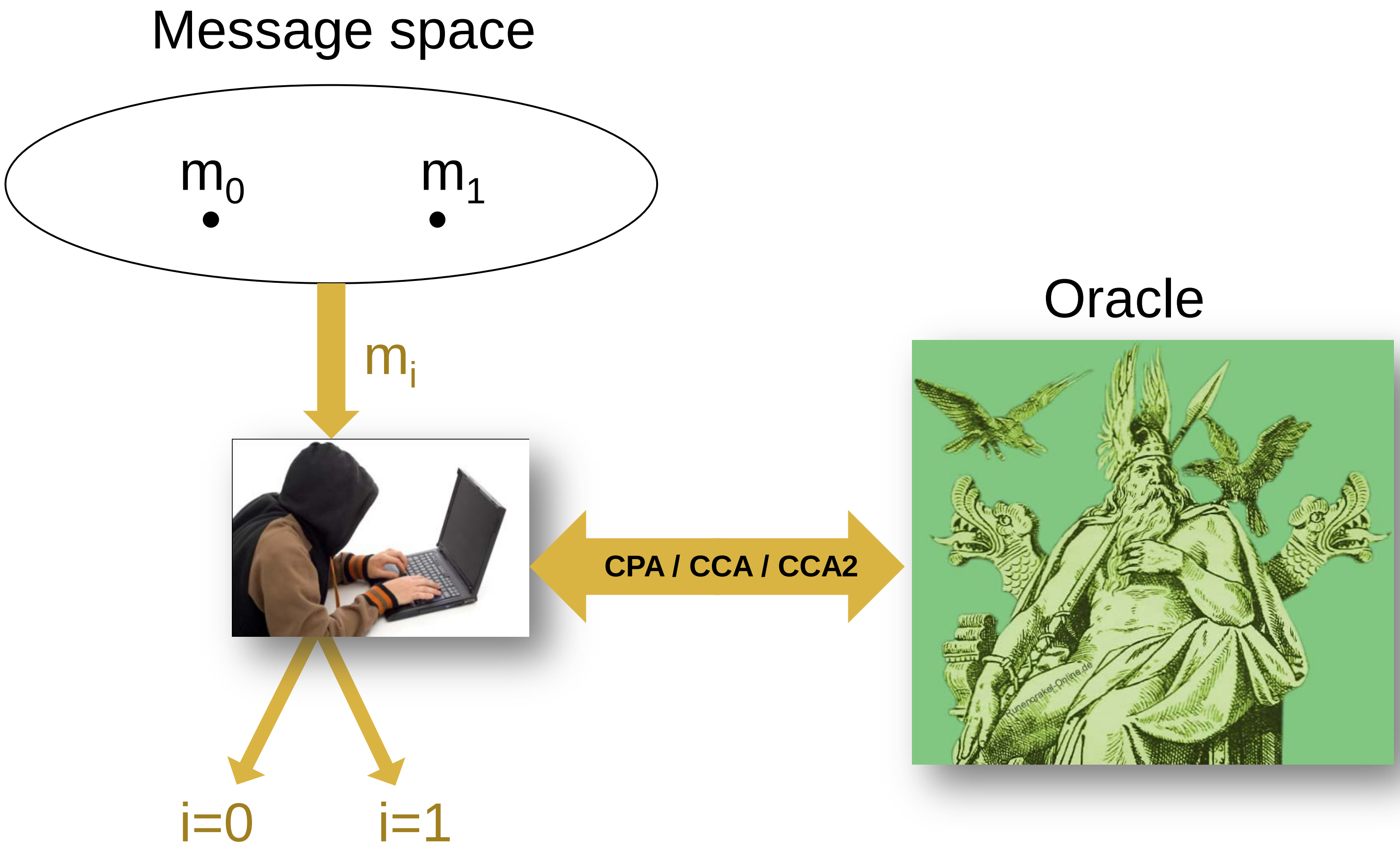
    → It is computationally infeasible for a passive adversary to derive significant information about a plaintext message from an observed ciphertext and a respective public (encryption) key
  - **Indistinguishability of ciphertexts** (aka **ciphertext indistinguishability**)

    → It is computationally infeasible for an adversary given the encryption of a message randomly chosen from a 2-element message space to identify the message choice with a probability significantly better than guessing (½)

    - … under CPA (IND-CPA) ⇔ Semantic security
    - … under CCA (IND-CCA)
    - … under CCA2 (IND-CCA2)

Asymmetric Encryption Systems

Introduction



Asymmetric Encryption Systems

Introduction

- **Nonmalleability (NM)**

  → It is computationally infeasible for an adversary to transform a ciphertext into another ciphertext which decrypts to a related plaintext, i.e., given a ciphertext c that is an encryption of some plaintext message m, it is infeasible to generate another ciphertext c' that decrypts to f(m), for a known function f, without necessarily knowing or learning m

  - … under CPA (NM-CPA)
  - … under CCA (NM-CCA)
  - … under CCA2 (NM-CCA2)

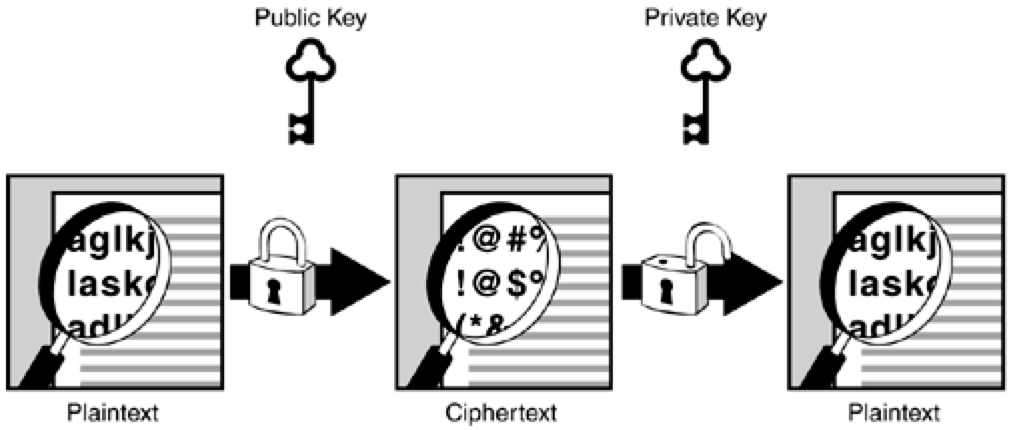
  It is known that NM-CPA $\Rightarrow$ IND-CPA (semantic security) and NM-CCA2 $\Leftrightarrow$ IND-CCA2

- IND-CCA2 (or NM-CCA2) is the preferred notion of security in the realm of asymmetric encryption

Asymmetric Encryption Systems

8.2 Basic Systems

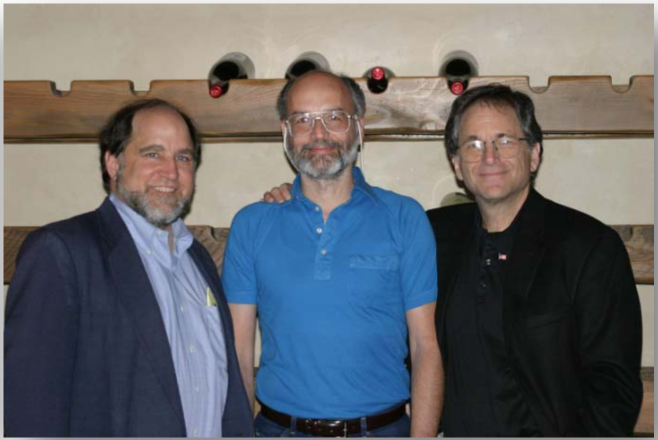
- RSA
- Rabin
- Elgamal



Asymmetric Encryption Systems

Basic Systems – RSA

- Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman (MIT, 1977)



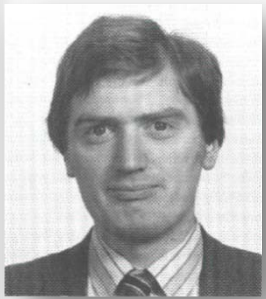
2002 Turing Award

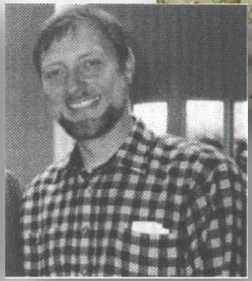
- Published in the *Communications of the ACM* (February 1978)
- U.S. patent 4,405,829 entitled „Cryptographic communications system and method" (filed on 12/14/1977, granted on 9/20/1983, and released to the public on 9/6/2000 – 2 weeks prior to expiration)
- The *Scientific American* publication of August 1977 (cf. RSA function) inhibited patents in almost all other countries

Asymmetric Encryption Systems

Basic Systems – RSA

- In 1997, it was publicly released that **non-secret encryption** had been proposed in an internal note at the British **Government Communications Headquarter (GCHQ)** in the early 1970s
- Non-secret encryption is similar to RSA (and Diffie-Hellman)
- The main inventor was Clifford Cocks (together with James H. Ellis and Malcolm J. Williamson)





Asymmetric Encryption Systems

Basic Systems – RSA

- The RSA public key cryptosystem is based on the RSA family of trapdoor permutations
- It yields an asymmetric encryption and a DSS
  - If the recipient‘s public key is used to encrypt a plaintext message, then RSA yields an asymmetric encryption system
    → the recipient‘s private key must be used to decrypt the ciphertext
  - If the sender‘s private key is used to encrypt a plaintext message (or hash value thereof), then RSA yields a DSS
    → the sender‘s public key must be used to verify the digital signature

Asymmetric Encryption Systems

Basic Systems – RSA

- The RSA Key Generation Algorithm **Generate(1^l)** is probabilistic and outputs a public key pair ((n,e),d)
- The algorithm operates in 2 steps
  - It randomly selects two l/2-bit prime numbers p and q, and computes the RSA modulus $n = pq$
  - It randomly selects an integer $1<e<\phi(n)$ with $\gcd(e,\phi(n)) = 1$, and computes another integer $1<d<\phi(n)$ with $de \equiv 1 \pmod{\phi(n)}$, using, for example, the extended Euclid algorithm, i.e., d represents the multiplicative inverse of e modulo $\phi(n)$
- Toy example
  - $p = 11, q = 23 \rightarrow n = 11 \cdot 23 = 253, \phi(n) = \phi(253) = 10 \cdot 22 = 220$
  - $e = 3$ (note that $\gcd(3,220) = 1$) $\rightarrow d = 147$ (note that $3 \cdot 147 = 441 \equiv 1 \pmod{220}$)

Asymmetric Encryption Systems

Basic Systems – RSA

- The RSA Encryption Algorithm **Encrypt(k,m)** is deterministic
- It takes as input a public key k = (n,e) and a plaintext message $m \in \mathbf{Z}_n$, and it generates as output the ciphertext

$$c = RSA_{n,e}(m) \equiv m^e \pmod{n}$$

- Toy example
  - m = 26
  - $c = 26^3 \pmod{253} \equiv 17{,}576 \pmod{253} = 119$

Asymmetric Encryption Systems

Basic Systems – RSA

- The RSA Decryption Algorithm **Decrypt(k^{-1},c)** is deterministic
- It takes as input a private key $k^{-1} = d$ and a ciphertext c, and it generates as output the original plaintext message

$$m = RSA_{n,d}(c) \equiv c^d \pmod{n}$$

- Toy example
  - $c = 119$
  - $m = 119^{147} \pmod{253} = 26$
- The efficiency of the RSA decryption algorithm can be improved using the **Chinese Reminder Theorem (CRT)**
- This requires the knowledge of the prime factorization of n (i.e., p and q)

Asymmetric Encryption Systems

Basic Systems – RSA

- The RSA asymmetric encryption system is correct (for all messages m with gcd(m,n) = 1)

$$
\begin{aligned}
m = RSA_{n,d}(c) &\equiv c^d \pmod{n} \\
&\equiv (m^e)^d \pmod{n} \\
&\equiv m^{ed} \pmod{n} \\
&\equiv m^{k\cdot\phi(n)+1} \pmod{n} \\
&\equiv \underbrace{m^{\phi(n)}\cdot\ldots\cdot m^{\phi(n)}}_{\text{k times}}\cdot m \pmod{n} \\
&\equiv \underbrace{1\cdot\ldots\cdot 1}_{\text{k times}}\cdot m \pmod{n} \\
&\equiv m \pmod{n}
\end{aligned}
$$

$ed \equiv 1 \pmod{\phi(n)} \Rightarrow ed \equiv k\cdot\phi(n)+1$

Fermat's Little Theorem (1607-1665):

p prime, $a \in \mathbf{Z} \Rightarrow a^{p-1} \equiv 1 \pmod{p}$

Euler's Theorem (1707-1783):

$a,n \in \mathbf{Z}$ with $\gcd(a,n) = 1 \Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$

- If gcd(m,n) ≠ 1, then the cottectness proof is more involved

Asymmetric Encryption Systems

Basic Systems – RSA

- Since its invention, the security of the RSA public key cryptosystem has been subject to a lot of public scrutiny
- It is known that RSAP $\leq_P$ IFP
- The converse is not known to be true (in the general case) and represents an open problem
- Problems that are computationally equivalent to the IFP
  - Compute $\phi(n)$ from n
  - Determine d from n and e (find the multiplicative inverse of e modulo n)
- The modulus n must be at least as large as to make it computa-tionally infeasible to factorize it (≥ 2,048 bits)
- Models and predictions about the future development of keylengths can be found at http://www.keylength.com

Asymmetric Encryption Systems

Basic Systems – RSA

- With regard to security, there are different criteria for public and private exponents
  - The public exponent can be arbitrarily small, e.g., $e = 2^{16} + 1 = 65{,}537$
  - The private exponent should not be too small, i.e., $d \geq n^{0.292}$ which is about 300 bits for an 1,024-bit modulus
- Also, the RSA asymmetric encryption system has the **bit security property** (i.e., an efficient algorithm for solving the RSAP can be constructed from an algorithm for predicting one single plaintext message bit)
- The bit security property is a double-edged sword
  - It provides evidence that all plaintext message bits are equally well protected
  - But it also provides a possibility to attack a „leaky“ implementation

Asymmetric Encryption Systems

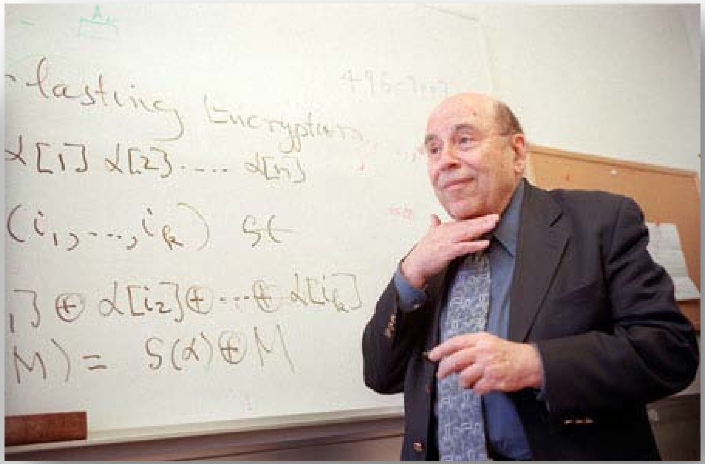
Basic Systems – RSA

- Several attacks are known and must be considered with care (in addition to side-channel attacks)
  - Common modulus attack (if n is reused)
  - CCA2 that exploit the multiplicative structure of the RSA function (i.e., the function is multiplicatively homomorphic)
    - To decrypt c, the adversary has the victim decrypt $c' \equiv cr^e \pmod{n}$ for some randomly chosen r
    - The victim computes $m' \equiv c'^d \equiv (cr^e)^d \equiv mr \pmod{n}$
    - The adversary computes $m \equiv m'/r \pmod{n}$
  - Low exponent attacks (if $m^e < n$)
- The RSA asymmetric encryption system should not be used natively
- Instead, messages should be preprocessed and encoded prior to applying the RSA function (e.g., OAEP)

Asymmetric Encryption Systems

Basic Systems – Rabin

- The RSAP is not known to be computationally equivalent to the IFP
- It is therefore theoretically possible to break the RSA public key cryptosystem without solving the IFP
- This possibility is worrisome, and – since the beginning of public key cryptography and the invention of RSA – people have been looking for public key cryptosystems for which breaking it is computationally equivalent to the IFP
- The first proposal was made in 1979 by Michael O. Rabin (1976 Turing Award)



Asymmetric Encryption Systems

Basic Systems – Rabin

- The Rabin asymmetric encryption system is based on the Square family of trapdoor (one-way) functions
- If the functions are restricted to QR_n, then the functions represent or permutations
- Breaking the Rabin system is computationally equivalent to solving the IFP
- This means
  - If somebody can solve the IFP (i.e., factorize n), then this person can break the Rabin system (the prime factors represent the private key)
  - If somebody can break the Rabin system, then this person can solve the IFP

Asymmetric Encryption Systems

Basic Systems – Rabin

- The Rabin Key Generation Algorithm **Generate(1^l)** takes as input a security parameter l (or 1^l, respectively) and generates as output a Blum integer n of this bit length
- More specifically, the Generate algorithm randomly selects two l/2-bit primes p and q (both equivalent to 3 modulo 4), and computes n = pq
  - n is the public key
  - (p,q) is the private key
- Toy example
  - p = 11, q = 23 (note that $11 \equiv 23 \equiv 3 \pmod 4$)
  - n = pq = 11·23 = 253

Asymmetric Encryption Systems

Basic Systems – Rabin

- Similar to the RSA asymmetric encryption system, the Rabin system can be used to encrypt and decrypt plaintext messages
- In the general case, these messages represent numbers and elements of $\mathbf{Z}_n = \{0,1,\ldots,n-1\}$
- The Rabin Encryption Algorithm **Encrypt(k,m)** is deterministic
- It takes as input a public key k = n and a plaintext message $m \in \mathbf{Z}_n$, and it generates as output the ciphertext

$$c = \text{Square}_n(m) \equiv m^2 \pmod{n}$$

- Toy example
  - $m = 158$
  - $c \equiv 158^2 \pmod{253} = 170$

Asymmetric Encryption Systems

Basic Systems – Rabin

- The Rabin Decryption Algorithm **Decrypt(k^{-1},c)** is deterministic
- It takes as input a private key k^{-1} = (p,q) and a ciphertext c, and it generates as output the square root of c modulo n representing m

$$m = Sqrt_n(c) \equiv c^{1/2} \pmod{n}$$

- Note that the recipient can find a square root of c modulo n iff he knows the prime factors of n (i.e., p and q)
- Also note that in the general case there is no single square root of c modulo n, but there are 4 of them (i.e., m_1,m_2,m_3, and m_4)
- The recipient must decide which m_i ($1 \leq i \leq 4$) represents the correct plaintext message (this ambiguity is a major disadvantage)
- In the toy example, the square roots of c = 170 modulo 253 are 26, 95, <u>158</u>, and 227

Asymmetric Encryption Systems

Basic Systems – Elgamal

- In 1984, Taher Elgamal found a way to turn the Diffie-Hellman key exchange protocol into a full-fledged public key crypto-system (i.e., there is an Elgamal asymmetric encryption system and an Elgamal DSS)

- Conceptually, the Elgamal public key cryptosystem works in any cyclic group in which the DLP is intractable (e.g., $\mathbf{Z}_p^*$)
- Breaking the Elgamal asymmetric encryption system can be shown to be computationally equivalent to solving the DHP

Asymmetric Encryption Systems

Basic Systems – Elgamal

- The Elgamal Key Generation Algorithm **Generate(1^l)** is the same as the one employed by the Diffie-Hellman key exchange protocol
- In the case of $\mathbf{Z}_p^*$, it generates an l-bit prime p and a generator g of $\mathbf{Z}_p^*$, i.e., $\langle g \rangle = \mathbf{Z}_p^*$ (p and g can be same for all users)
- For each user, the algorithm then randomly selects a private key $x \in \mathbf{Z}_p^*$ and computes the respective public key $y \equiv g^x \pmod{p}$
- Toy example
  - $p = 17$, $g = 7$ (note that $\langle 7 \rangle = \mathbf{Z}_{17}^*$, since $7^1 = 7$, $7^2 = 15$, $7^3 = 3$, $7^4 = 4$, $7^5 = 11$, $7^6 = 9$, $7^7 = 12$, $7^8 = 16$, $7^9 = 10$, $7^{10} = 2$, $7^{11} = 14$, $7^{12} = 13$, $7^{13} = 6$, $7^{14} = 8$, $7^{15} = 5$, and $7^{16} = 1$)
  - Private key is $x = 6$
  - Public key is $y \equiv 7^6 \pmod{17} \equiv 117{,}649 \pmod{17} = 9$

Asymmetric Encryption Systems

Basic Systems – Elgamal

- The Elgamal Encryption Algorithm **Encrypt(k,m)** is probabilistic
- It takes as input a public key k = (p,g,y) and a plaintext message $m \in \mathbf{Z}_p$, and it generates as output the ciphertext $c = (c_1,c_2)$ that is computed as follows:

$$\overset{\xleftrightarrow{k}}{(p,g,y,m)}$$

$r \in_R \mathbf{Z}_p^*$

$K \equiv y^r \pmod{p}$

$c_1 \equiv g^r \pmod{p}$

$c_2 \equiv Km \pmod{p}$

(c_1,c_2)

- Toy example (m = 7)
  - r = 3
  - $K \equiv 9^3 \pmod{17} = 15$
  - $c_1 \equiv 7^3 \pmod{17} = 3$
  - $c_2 \equiv 15 \cdot 7 \pmod{17} = 3$
- Hence, 7 is encrypted as (3,3)

Asymmetric Encryption Systems

Basic Systems – Elgamal

- The Elgamal Decryption Algorithm **Decrypt(k^{-1},c)** is deterministic
- It takes as input a private key $k^{-1} = (p,q)$ and a ciphertext c, and it generates as output the original plaintext message that is computed as follows:

$$
\begin{array}{l}
k^{-1} \\
\downarrow \\
(x,c) \\
\hline
K \equiv c_1^x \pmod{p} \\
m \equiv c_2/K \equiv c_2 K^{-1} \pmod{p} \\
\hline
(m)
\end{array}
$$

- Toy example $(c_1,c_2) = (3,3)$)
  - $K \equiv 3^6 \pmod{17} = 15$
  - $K^{-1} \pmod{17} = 8$ ($15 \cdot 8 = 120 \equiv 1 \pmod{17}$)
  - $m \equiv 3 \cdot 8 \pmod{17} = 7$

- Note that $c_1^x \equiv (g^r)^x \equiv (g^x)^r \equiv y^r \equiv K \pmod{p}$

Asymmetric Encryption Systems

Basic Systems – Elgamal

- The Elgamal asymmetric encryption system is semantically secure and provides IND-CPA
- But it is highly malleable and does not provide IND-CCA
- For example, given a ciphertext (c_1,c_2) of some (possibly unknown) plaintext message m, $(c_1,2c_2)$ represents the ciphertext for the plaintext message 2m
- Whether this fact poses a problem depends on the application context
- Many other asymmetric encryption systems based on Elgamal have been proposed (e.g., Paillier, Cramer-Shoup, ...)

Asymmetric Encryption Systems

8.3 Secure Systems

- There are a few asymmetric encryption systems known to be semantically secure or to provide IND-CCA2
  - Probablistic encryption is semantically secure (proof in the standard model)
  - Optimal Asymmetric Encryption Padding (OAEP) provides IND-CCA2 (proof in the random oracle model)
  - Cramer-Shoup provides IND-CCA2 (proof in the standard model)

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- **Probablistic encryption** was proposed by Shafi Goldwasser and Silvio Micali in the early 1980s
- It is based on the **Quadratic Residue Problem (QRP)**, i.e., for $n \in \mathbf{N}$ and $x \in \mathbf{Z}_n^*$, decide whether x is a square or quadratic residue (i.e., $x \in QR_n$), where

$$QR_n := \{x \in \mathbf{Z}_n^* \mid \exists y \in \mathbf{Z}_n^* : y^2 \equiv x \pmod{n}\}$$

- Unless the factorization of n is known, it is not known how to efficiently solve the QRP
- It is widely believed that the QRP is computationally equivalent to the IFP (i.e., $QRP \equiv_P IFP$)

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- Mathematical preliminaries
  - For every integer x and odd prime p, the **Legendre symbol (x|p)** is defined as follows:

$$(x|p) = \begin{cases} 0 & \text{if } x \equiv 0 \pmod{p} \\ +1 & \text{if } x \in QR_p \\ -1 & \text{if } x \in QNR_p \end{cases}$$

  - It can be computed efficiently using $(x|p) = x^{(p-1)/2} \pmod{p}$
  - For every prime p, $x \in QR_p \Leftrightarrow (x|p) = 1$
  - This means that the Legendre symbol (x|p) is 1 iff x is a quadratic residue modulo p

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- The **Jacobi symbol (x|n)** is a generalization of the Legendre symbol (for any composite integer n)
- If the prime factorization of n is known, then (x|n) can be computed efficiently (as the product of the Legendre symbols (x|p) of the respective prime factors of n)

$$\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right)^{\alpha_1}\left(\frac{x}{p_2}\right)^{\alpha_2} \cdots \left(\frac{x}{p_k}\right)^{\alpha_k}$$

- Unlike (x|p), for every composite number n

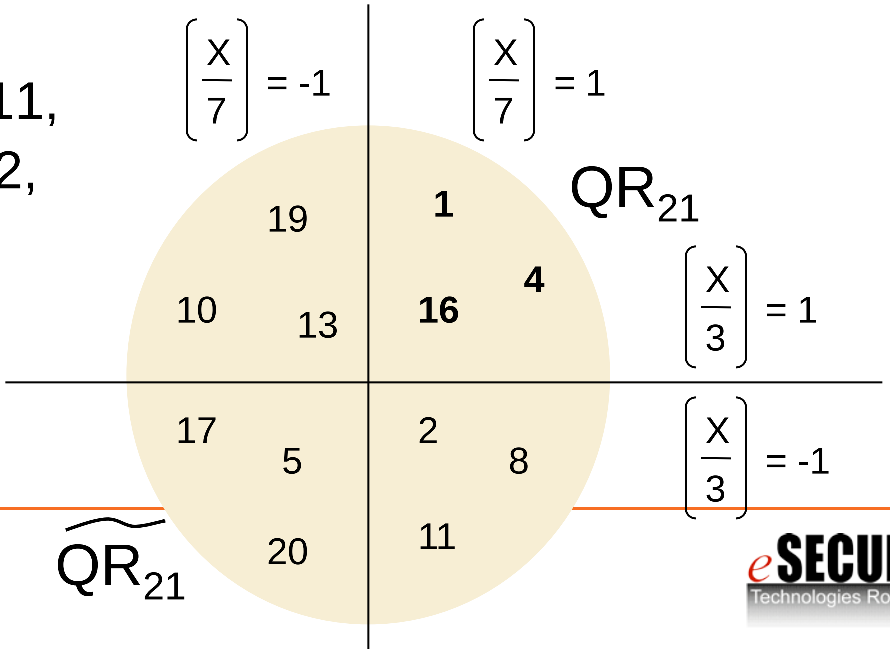
$$x \in QR_n \Rightarrow (x|n) = 1$$
$$x \in QR_n \not\Leftarrow (x|n) = 1$$

- If x is a quadratic residue modulo n, then the Jacobi symbol (x|n) is 1
- The converse need not be true, i.e., if the Jacobi symbol (x|n) is 1, then x need not be a quadratic residue modulo n

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- If $(x|n) = -1$, then x is a quadratic nonresidue modulo n, i.e., $(x|n) = -1 \Rightarrow x \in QNR_n$
- J_n refers to the set of all elements of $\mathbf{Z}_n^*$ with Jacobi symbol 1, i.e., $J_n = \{x \in \mathbf{Z}_n^* \mid (x|n) = 1\}$
- $\widetilde{QR}_n = J_n \setminus QR_n$ refers to the set of all pseudosquares modulo n
- For $n = pq$, $|QR_n| = |\widetilde{QR}_n| = (p-1)(q-1)/4$
- Hence, half of the elements of J_n are squares and the other half are pseudosquares modulo n
- Example: $\mathbf{Z}_{21}^* = \{1,2,4,5,8,10,11,13,16,17,19,20\}$ with $|\mathbf{Z}_{21}^*| = 12$, $J_n = \{1,4,516,17,20\}$, $QR_{21} = \{1,4,16\}$, and $\widetilde{QR}_{21} = \{5,17,20\}$



$\left(\frac{x}{7}\right) = -1$
$\left(\frac{x}{7}\right) = 1$
QR_{21}
$\left(\frac{x}{3}\right) = 1$
$\left(\frac{x}{3}\right) = -1$
$\widetilde{QR}_{21}$
19
10
13
1
4
16
17
5
20
2
8
11

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- Probabilistic encryption exploits the fact that for an arbitrary element of J_n it is computationally intractable to decide whether it is a quadratic residue (i.e., square) or a pseudosquare modulo n
- The Key Generation Algorithm **Generate(1^l)** takes as input a security parameter l and generates as output two primes p and q and a modulus n = pq of l bits
- In addition, the algorithm selects a pseudosquare $y \in \widetilde{QR}_n$
- (n,y) is the public key and (p,q) is the private key

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- As its name suggests, the Encryption Algorithm **Encrypt(k,m)** is probabilistic
- It takes as input a public key k = (n,y) and a message m, and it generates as output a ciphertext c
- For every message bit m_i, the Encrypt algorithm chooses $x_i \in_R \mathbf{Z}_n^*$ and computes

$$c_i = \begin{cases} x_i^2 \pmod{n} & \text{if } m_i = 0 \\ yx_i^2 \pmod{n} & \text{if } m_i = 1 \end{cases}$$

c_i is a square modulo n

c_i is a pseudosquare modulo n

- Every message bit m_i is encrypted with an element $c_i \in \mathbf{Z}_n^*$
- Hence, the resulting ciphertext is the k-tuple $c = (c_1,\ldots,c_k)$ which is k·l bits (this is prohibitively inefficient)

Asymmetric Encryption Systems

Secure Systems – Probabilistic Encryption

- The Decryption Algorithm **Decrypt(k^{-1},c)** is deterministic
- It takes as input a private key $k^{-1} = (p,q)$ and a ciphertext $c = (c_1,\ldots,c_k)$, and it generates as output the k-bit plaintext message m
- Again, the algorithm proceeds sequentially on each ciphertext element c_i $(i = 1,\ldots,k)$
- If $c_i \in QR_n$, then $m_i = 0$ (otherwise, $c_i \in \widetilde{QR}_n$ and $m_i = 1$)

$$m_i = \begin{cases} 0 & \text{if } (c_i|p) = 1 \\ 1 & \text{otherwise} \end{cases}$$

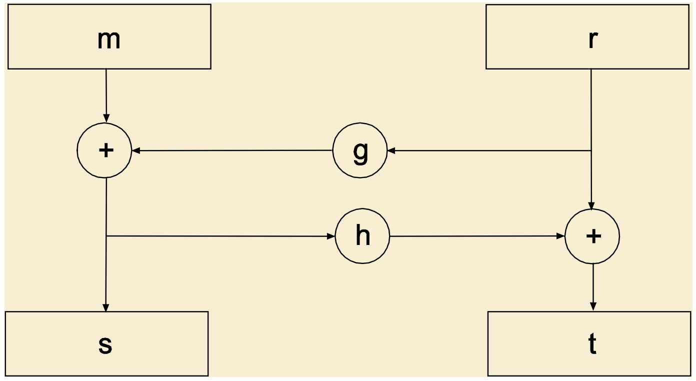
- The plaintext message is $m = m_1\ldots m_k$

Asymmetric Encryption Systems

Secure Systems – OAEP

- OAEP was developed and proposed by Mihir Bellare und Phil Rogaway
- It is basically a padding scheme used prior to encryption to provide IND-CCA2 (e.g., RSA-OAEP) − The proof is in the random oracle model

$$\text{OAEP}(m) = (s,t) = \underbrace{m \oplus g(r)}_{s} \,||\, \underbrace{r \oplus h(m \oplus g(r))}_{t}$$



Asymmetric Encryption Systems

Secure Systems – OAEP

- To decrypt a message encrypted with RSA-OAEP, the recipient must first decrypt the message (using the RSA Decrypt algorithm) to get OAEP(m) = (s,t)
- He or she must then extract m and r from (s,t)

$$\begin{aligned} t \oplus h(s) &= r \oplus h(m \oplus g(r)) \oplus h(m \oplus g(r)) \\ &= r \\ s \oplus g(r) &= m \oplus g(r) \oplus g(r)) \\ &= m \end{aligned}$$

- RSA-OAEP is used in PKCS #1

Asymmetric Encryption Systems

Secure Systems – Cramer-Shoup

- In 1998, Ronald Cramer and Victor Shoup proposed an asymmetric encryption system that provides IND-CCA2 (nonmalleable)
- The security proof is in the standard model
- The Cramer-Shoup asymmetric encryption system enhances the Elgamal asymmetric encryption system (its security is based on the DDHP)
- The distinguishing feature of the Cramer-Shoup asymmetric encryption system is its efficiency

Asymmetric Encryption Systems

Secure Systems – Cramer-Shoup

- The Cramer-Shoup Key Generation Algorithm **Generate(1^l)** takes as input a security parameter l and generates as output a public key pair
  - It selects a cyclic group G of order q (with bit-size l) and 2 generators g_1 and g_2, i.e., G = {0,…,q-1} (G, q, g_1 and g_2 are publicly known)
  - It randomly selects 6 values (x_1,x_2,y_1,y_2,z_1,z_2) from G (this 6-tuple represents the private key)
  - It computes $d = g_1^{x_1} g_2^{x_2}$

    $e = g_1^{y_1} g_2^{y_2}$

    $f = g_1^{z_1} g_2^{z_2}$
- (G,q, g_1,g_2,d,e,f) represents the public key

Asymmetric Encryption Systems

Secure Systems – Cramer-Shoup

- The Cramer-Shoup Encryption Algorithm **Encrypt(k,m)** is probabilistic
- It takes as input a public key k = (G,q, g_1,g_2,d,e,f) and a plaintext message m, and it generates as output a respective ciphertext c

(G,q,g_1,g_2,d,e,f,m)

$r \in_R \{0,\ldots,q-1\}$

$u_1 = g_1^r$

$u_2 = g_2^r$

$w = f^r m$

$h = H(u_1,u_2,w)$

$v = d^r e^{rh}$

(u_1,u_2,w,v)

H refers to a cryptographic (i.e., collision-resistant) hash function

Asymmetric Encryption Systems

Secure Systems – Cramer-Shoup

- The Cramer-Shoup Encryption Decryption **Decrypt(k^{-1},c)** is deterministic
- It takes as input a private key $k^{-1} = (x_1,x_2,y_1,y_2,z_1,z_2)$ and a cipher-text $c = (u_1,u_2,w,v)$, and it generates as output a respective plain-text message m

$$(x_1,x_2,y_1,y_2,z_1,z_2,u_1,u_2,w,v)$$

$$h = H(u_1,u_2,w)$$

$$\text{If } (u_1^{x_1}u_2^{x_2}(u_1^{y_1}u_2^{y_2})^h = v) \text{ then } m = w/(u_1^{z_1}u_2^{z_2})$$

else abort

$$(m)$$

Asymmetric Encryption Systems

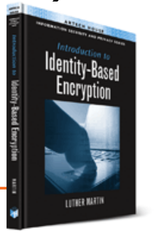
8.4 Identity-Based Encryption

- In asymmetric system, every user entity have a public key pair
- The public keys look arbitrary and random, hence one cannot easily assign a public key to a specific entity
- In 1978, Loren M. Kohnfelder (MIT) proposed the notion of a public **key certificate** to address the key assignment problem
- A public key certificate is a data structure that
  - is issued by a trusted (or trustworthy) entity – called **certification authority (CA)** or **certification service provider (CSP)**
  - claims that a specific public key belongs to a specific entity
  - is digitally signed by the certificate-issuing CA/CSP
- If there are multiple (mutually trusting) CAs in place, then one talks about a **public key infrastructure (PKI)**

Asymmetric Encryption Systems

Identity-Based Encryption

- In the early 1980s, Adi Shamir came up with the notion of **identity-based encryption (IBE)**
- The public key uniquely identifies the key holder → one has no longer to care about the ownership of public keys, public key certificates, and PKIs
  - The main advantage is that neither public key certificates nor mechanisms to distribute them (e.g., directory services) are needed
  - The main disadvantage is that a trusted (central) authority is needed to generate public key pairs and distribute them to the appropriate entities
- In 2001, IBE systems were proposed by Dan Boneh and Matthew Franklin (based on bilinear pairings on elliptic curves) and Clifford Cocks (based on quadratic residues)
- The usefulness of IBE is controversially discussed



Asymmetric Encryption Systems

8.5 Homomorphic Encryption

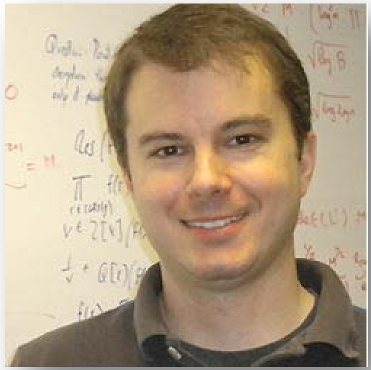
- In 1978, Ron Rivest, Len Adleman, and Michael Dertouzos introdu-ced the notion of a **homomorphic encryp-tion**, i.e., to encrypt data in a way that allows computations to be done only on the cipher-texts (i.e., without decryption)
- Homomorphic encryption has interesting applications in the realm of outsourcing and cloud computing

$$\begin{tikzcd}
m_1,m_2 \arrow[r, "\circledcirc"] \arrow[dd] & m_1 \circledcirc m_2 \arrow[d] \\
 & E(m_1 \circledcirc m_2) \arrow[d, equal] \\
E(m_1),E(m_2) \arrow[r, "\odot"] & E(m_1) \odot E(m_2)
\end{tikzcd}$$

Asymmetric Encryption Systems

Homomorphic Encryption

- Many asymmetric encryption systems in use today are partially homomorphic (+ or ·)
  - (Unpadded) RSA (·)
  - Elgamal (·), Paillier (+)
  - Probabilistic encryption (·)
  - ...



- Until 2009, it was not clear whether **fully homomorphic encryp-tion (FHE)** is feasible (+ and ·)
- In 2009, Craig Gentry solved the problem and proposed a FHE system using latice-based cryptography
- This system has been improved and many researchers have started to work on FHE systems

Asymmetric Encryption Systems

8.6 Final Remarks

- In addition to the asymmetric encryption systems addressed so far, there are other systems that have been proposed in the past
- Some of these systems have been broken and become obsolete
- For example, the NP-complete subset sum problem has served as a basis for many public key cryptosystems (e.g., Merkle-Hellman)
- All knapsack-based public key cryptosystems have been broken, including the Chor-Rivest knapsack cryptosystem
- Knapsack-based cryptosystems illustrate that it is necessary but usually not sufficient that a public key cryptosystem is based on a mathematically hard problem
- Breaking a knapsack-based public key cryptosystem is possible without solving the subset sum problem

Asymmetric Encryption Systems

Final Remarks

- There are still a few asymmetric encryption systems that have turned out to be resiatnt against cryptanalytical attacks (e.g., McEliece)
- The assumption that public keys are published in certified form raises questions
  - How does one ensure that all entities have public keys?
  - How does one publish them?
  - How does one certify them?
  - How does one handle the revocation of public keys?
  - ….

- The operation of a PKI is heavily involved
- It represents the Achilles heel of public key cryptography

9 Cryptographic Hash Functions

Cryptographic Hash Functions

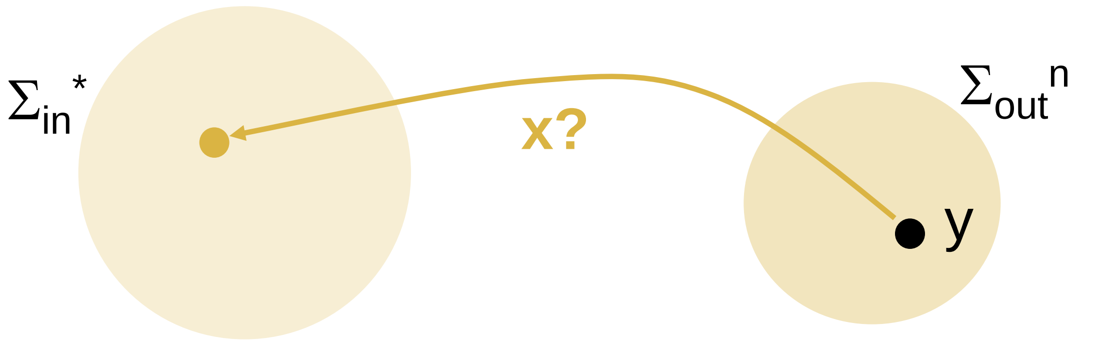
9.1 Introduction

- A **hash function** is an efficiently computable function h: $\Sigma_{in}^* \rightarrow \Sigma_{out}^n$ (or h: $\Sigma_{in}^{n_{max}} \rightarrow \Sigma_{out}^n$, repectively)
- Σ_{in} and Σ_{out} are the input and output alphabets, typically the binary alphabet $\Sigma = \{0,1\}$
- As mentioned before, a cryptographic hash function must be
  - Preimage resistant (one-way)
  - Second-preimage resistant (weakly collision resistant)
  - Collision resistant (strongly collision resistant)

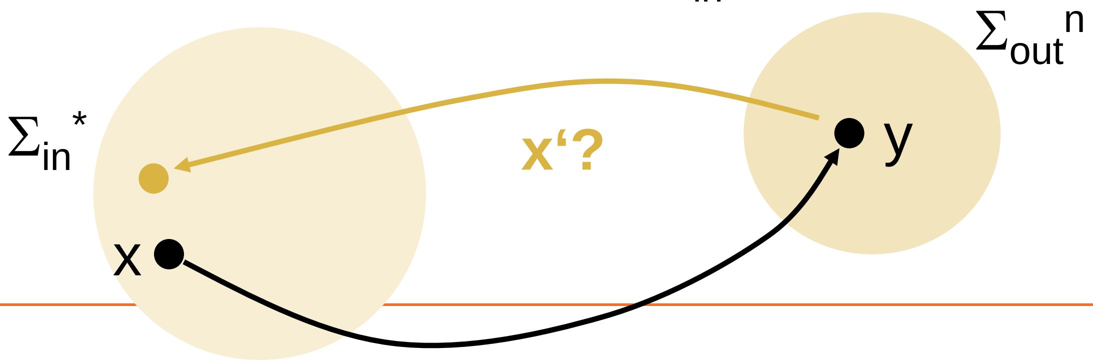
Cryptographic Hash Functions

Introduction

- A hash function h is **preimage resistant** (or **one-way**) if it is computationally infeasible to find an $x \in \Sigma_{in}^*$ with $h(x) = y$ for a $y \in_R \Sigma_{out}^n$



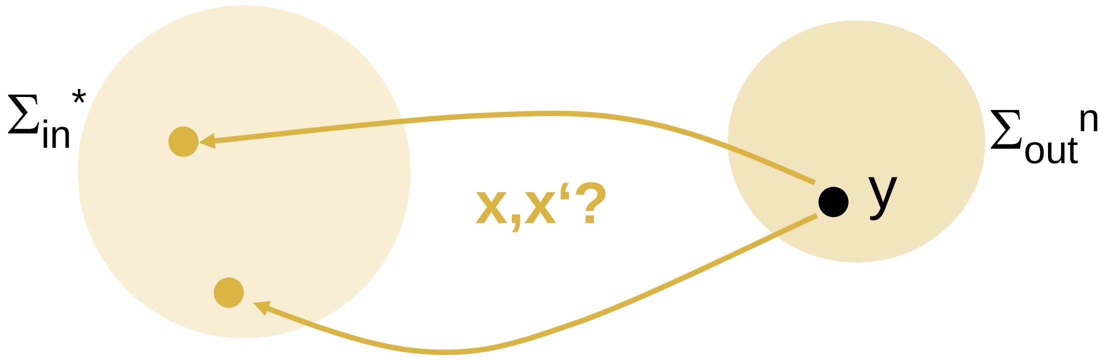
- A hash function h is **second-preimage resistant** (or **weakly collision resistant**) if it is computationally infeasible to find an $x' \in \Sigma_{in}^*$ with $x' \neq x$ and $h(x') = h(x)$ for an $x \in \Sigma_{in}^*$



Cryptographic Hash Functions

Introduction

- A hash function h is **collision resistant** (or **strongly collision resistant**) if it is computationally infeasible to find $x,x' \in \Sigma_{in}^{*}$ with $x' \neq x$ and $h(x') = h(x)$



Cryptographic Hash Functions

Introduction

- Exercise 9-1: Birthday paradox

  1. How many persons are required (e.g., in a room) such that the probability that at least one person has a given birthday is at least ½?
  2. How many persons are required such that the probability that at least two persons have the same birthday is at least ½?

Cryptographic Hash Functions

Introduction

- Collision resistance is a stronger requirement than second-preimage resistance (due to the birthday paradox)
- But preimage resistance is an inherently different requirement
- A collision resistant hash function need not be preimage resistant
- For example, let g be a collision resistant hash function with an n-bit output and h a pathological (n+1)-bit hash function

$$h(x) = \begin{cases} 1 \,||\, x & \text{if } |x|=n \\ 0 \,||\, g(x) & \text{otherwise} \end{cases}$$

- h is still collision resistant but not preimage resistant
- For all h(x) that begin with a 1, a primage can be trivially computed (by removing the 1)

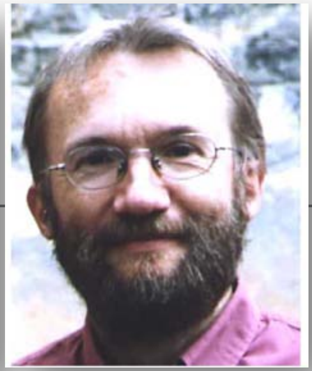
Cryptographic Hash Functions

Introduction

- Consequently, there are 2 types of cryptographic hash functions
  - One-way hash functions (OWHFs)
    - Preimage resistant
    - Second-preimage resistant (weakly collision resistant)
  - Collision resistant hash functions (CRHFs)
    - Preimage resistant
    - Collision resistant (strongly collision resistant)

Cryptographic Hash Functions

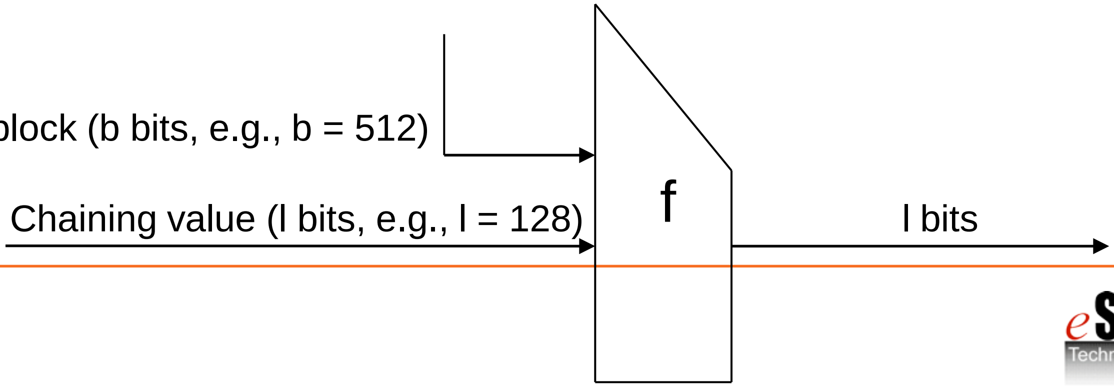
9.2 Merkle-Damgård Construction



- Most cryptographic hash functions in use today follow a construction that was independently proposed by Ralph Merkle and Ivan Damgård in the late 1980s
- The **Merkle-Damgård construction** employs a collision resistant compression function

$$f: \Sigma^{l+b} \rightarrow \Sigma^{l} \quad (\text{with } l,b \in \mathbf{N})$$

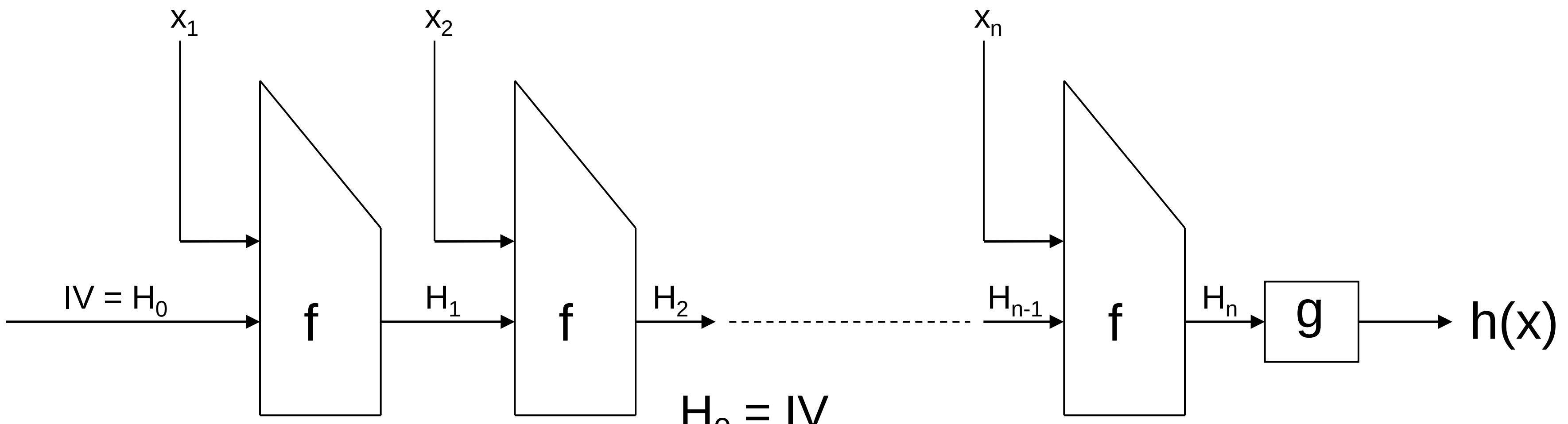
that is applied iteratively to successive b-bit blocks $x_1,\ldots,x_n$ of message x


Message block (b bits, e.g., b = 512)
Chaining value (l bits, e.g., l = 128)
f
l bits

Cryptographic Hash Functions

Merkle-Damgård Construction

- A cryptographic hash function that follows the Merkle-Damgård construction is also called **iterated hash function**
- An iterated hash function h inherits the collision resistance property from the underlying compression function f



$H_0 = IV$

$H_i = f(H_{i-1},x_i)$ for $i = 1,\ldots,n$

$h(x) = g(H_n)$

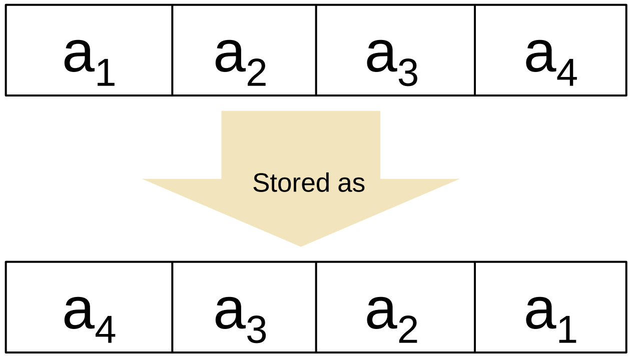
Cryptographic Hash Functions
9.3 Exemplary Hash Functions

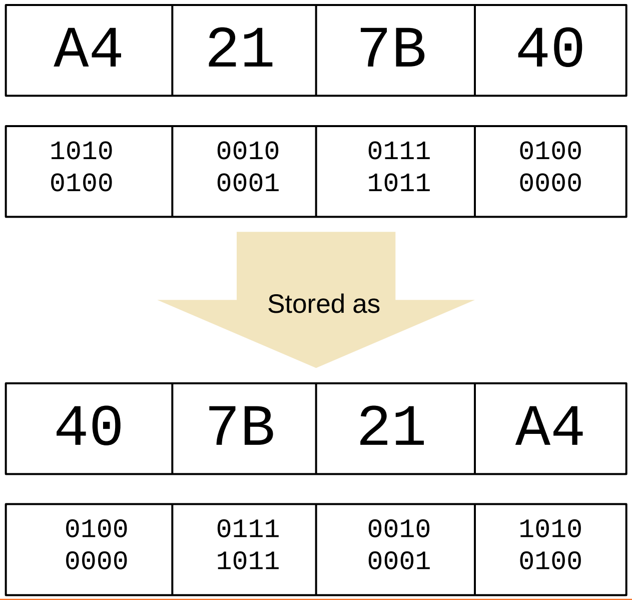
- MD2 (RFC 1319, 1992)
- MD4 (RFC 1320, 1992)
- MD5 (RFC 1321, 1992)
- SHA-1 (FIPS PUB 180-1, 1995)
- SHA-2
  - SHA-256, SHA-384, and SHA-512 (FIPS PUB 180-2, 2002)
  - SHA-224 (FIPS PUB 180-2 change note, 2004)
- RIPEMD-128 and RIPEMD-160
- …

Cryptographic Hash Functions

Exemplary Hash Functions – MD4

- MD4 implements a Merkle-Damgård construction with b = 512 and l = 128
- It was designed (and optimized) for 32-bit processors with little-endian architecture, i.e., a 4-byte word $a_1a_2a_3a_4$ is stored as $a_4a_3a_2a_1$, and hence it represents $a_4 2^{24} a_3 2^{16} a_2 2^{8} a_1$





Cryptographic Hash Functions

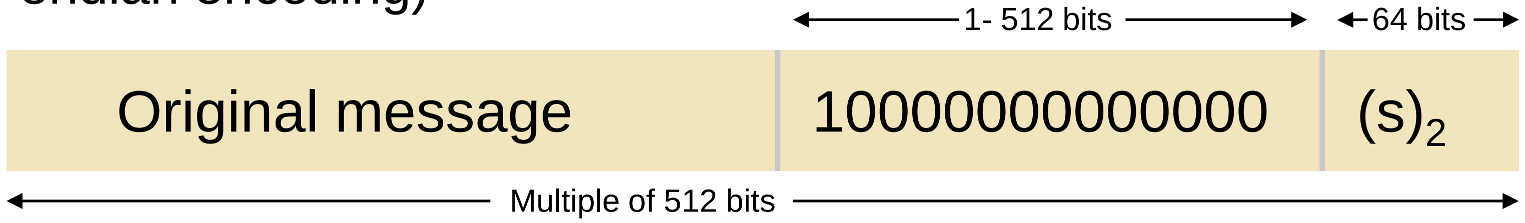
Exemplary Hash Functions – MD4

- Let $m = m_0 m_1 \ldots m_{s-1}$ be an s-bit message to be hashed
- First, an array

$$M = M[0]M[1]\ldots M[N-1]$$

is constructed, where M[i] for i = 0,…,N-1 represents a 32-bit word and $N \equiv 0 \bmod 16 \rightarrow |M|$ is a multiple of $32 \cdot 16 = 512$ bits

- M is constructed in 2 steps
  - The message m is padded so that its bit length is congruent to 448 modulo 512, i.e., $|m| \equiv 448 \pmod{512}$
  - A 64-bit binary representation of s is appended to the message (little-endian encoding)



Cryptographic Hash Functions

Exemplary Hash Functions – MD4

- MD4 algorithm (overview)
- A, B, C, and D represent 32 bit (4 byte) registers

Iteration of the compression function

```
(m = m_0 m_1 … m_{s-1})
--------------------------------
Construct M = M[0]M[1]…M[N-1]
A ← 0x67452301
B ← 0xEFCDAB89
C ← 0x98BADCFE
D ← 0x10325476
for i = 0 to N/16 do
  for j = 0 to 15 do X[j] = M[i·16+j]
  A‘ ← A
  B‘ ← B
  C‘ ← C
  D‘ ← D
  Round 1
  Round 2
  Round 3
  A ← A + A‘ (addition is modulo 2^32)
  B ← B + B‘
  C ← C + C‘
  D ← D + D‘
--------------------------------
(h(m) = A || B || C || D)
```

Cryptographic Hash Functions

Exemplary Hash Functions – MD4

- Building blocks for the 3 round functions of MD4
  - Boolean operations

$X \wedge Y$	AND (bitwise and of X and Y)
$X \vee Y$	OR (bitwise or of X and Y)
$X \oplus Y$	XOR (bitwise exclusive or of X and Y)
$X \neg Y$	NOT (bitwise complement of X)
$X + Y$	Integer addition of X and Y modulo 232
$X \hookleftarrow s$	Circular left shift of X by s positions ($0 \leq s \leq 31$)

  - Functions
    - $f(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$
    - $g(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$
    - $h(X,Y,Z) = X \oplus Y \oplus Z$
  - Constants
    - $c_1 = \lfloor 2^{30} \cdot 2^{1/2} \rfloor =$ `0x5A827999`
    - $c_2 = \lfloor 2^{30} \cdot 3^{1/2} \rfloor =$ `0x6ED9EBA1`

Cryptographic Hash Functions

Exemplary Hash Functions – MD4

Round 1

1. $A \leftarrow (A + f(B,C,D) + X[0]) \hookleftarrow 3$
2. $D \leftarrow (D + f(A,B,C) + X[1]) \hookleftarrow 7$
3. $C \leftarrow (C + f(D,A,B) + X[2]) \hookleftarrow 11$
4. $B \leftarrow (B + f(C,D,A) + X[3]) \hookleftarrow 19$
5. $A \leftarrow (A + f(B,C,D) + X[4]) \hookleftarrow 3$
6. $D \leftarrow (D + f(A,B,C) + X[5]) \hookleftarrow 7$
7. $C \leftarrow (C + f(D,A,B) + X[6]) \hookleftarrow 11$
8. $B \leftarrow (B + f(C,D,A) + X[7]) \hookleftarrow 19$
9. $A \leftarrow (A + f(B,C,D) + X[8]) \hookleftarrow 3$
10. $D \leftarrow (D + f(A,B,C) + X[9]) \hookleftarrow 7$
11. $C \leftarrow (C + f(D,A,B) + X[10]) \hookleftarrow 11$
12. $B \leftarrow (B + f(C,D,A) + X[11]) \hookleftarrow 19$
13. $A \leftarrow (A + f(B,C,D) + X[12]) \hookleftarrow 3$
14. $D \leftarrow (D + f(A,B,C) + X[13]) \hookleftarrow 7$
15. $C \leftarrow (C + f(D,A,B) + X[14]) \hookleftarrow 11$
16. $B \leftarrow (B + f(C,D,A) + X[15]) \hookleftarrow 19$

Round 2

1. $A \leftarrow (A + g(B,C,D) + X[0] + c_1) \hookleftarrow 3$
2. $D \leftarrow (D + g(A,B,C) + X[4] + c_1) \hookleftarrow 5$
3. $C \leftarrow (C + g(D,A,B) + X[8] + c_1) \hookleftarrow 9$
4. $B \leftarrow (B + g(C,D,A) + X[12] + c_1) \hookleftarrow 13$
5. $A \leftarrow (A + g(B,C,D) + X[1] + c_1) \hookleftarrow 3$
6. $D \leftarrow (D + g(A,B,C) + X[5] + c_1) \hookleftarrow 5$
7. $C \leftarrow (C + g(D,A,B) + X[9] + c_1) \hookleftarrow 9$
8. $B \leftarrow (B + g(C,D,A) + X[13] + c_1) \hookleftarrow 13$
9. $A \leftarrow (A + g(B,C,D) + X[2] + c_1) \hookleftarrow 3$
10. $D \leftarrow (D + g(A,B,C) + X[6] + c_1) \hookleftarrow 5$
11. $C \leftarrow (C + g(D,A,B) + X[10] + c_1) \hookleftarrow 9$
12. $B \leftarrow (B + g(C,D,A) + X[14] + c_1) \hookleftarrow 13$
13. $A \leftarrow (A + g(B,C,D) + X[3] + c_1) \hookleftarrow 3$
14. $D \leftarrow (D + g(A,B,C) + X[7] + c_1) \hookleftarrow 5$
15. $C \leftarrow (C + g(D,A,B) + X[11] + c_1) \hookleftarrow 9$
16. $B \leftarrow (B + g(C,D,A) + X[15] + c_1) \hookleftarrow 13$

Cryptographic Hash Functions

Exemplary Hash Functions – MD4

Round 3

1. $A \leftarrow (A + h(B,C,D) + X[0] + c_2) \hookleftarrow 3$
2. $D \leftarrow (D + h(A,B,C) + X[8] + c_2) \hookleftarrow 9$
3. $C \leftarrow (C + h(D,A,B) + X[4] + c_2) \hookleftarrow 11$
4. $B \leftarrow (B + h(C,D,A) + X[12] + c_2) \hookleftarrow 15$
5. $A \leftarrow (A + h(B,C,D) + X[2] + c_2) \hookleftarrow 3$
6. $D \leftarrow (D + h(A,B,C) + X[10] + c_2) \hookleftarrow 9$
7. $C \leftarrow (C + h(D,A,B) + X[6] + c_2) \hookleftarrow 11$
8. $B \leftarrow (B + h(C,D,A) + X[14] + c_2) \hookleftarrow 15$
9. $A \leftarrow (A + h(B,C,D) + X[1] + c_2) \hookleftarrow 3$
10. $D \leftarrow (D + h(A,B,C) + X[9] + c_2) \hookleftarrow 9$
11. $C \leftarrow (C + h(D,A,B) + X[5] + c_2) \hookleftarrow 11$
12. $B \leftarrow (B + h(C,D,A) + X[13] + c_2) \hookleftarrow 15$
13. $A \leftarrow (A + h(B,C,D) + X[3] + c_2) \hookleftarrow 3$
14. $D \leftarrow (D + h(A,B,C) + X[11] + c_2) \hookleftarrow 9$
15. $C \leftarrow (C + h(D,A,B) + X[7] + c_2) \hookleftarrow 11$
16. $B \leftarrow (B + h(C,D,A) + X[15] + c_2) \hookleftarrow 15$

Due to some early found weaknesses and vulnerabilities, MD4 has never been widely deployed in practice

Cryptographic Hash Functions

Exemplary Hash Functions – MD5

- MD5 is a strengthened version of MD4
- It is conceptually and structurally similar to MD4 (Merkle-Damgård construction with $b = 512$ and $l = 128$)
- Major differences
  - 4 rounds (instead of 3)
  - Modified function $g(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$
  - New function $i(X,Y,Z) = Y \oplus (X \vee (\neg Z))$
  - 64-element table T constructed from the sine function (where i is taken in radians)

$$T[i] = \lfloor 4{,}294{,}967{,}296 \cdot |\sin(i)| \rfloor$$

Cryptographic Hash Functions

Exemplary Hash Functions – MD5

- MD5 algorithm (overview)

```
(m = m_0 m_1 … m_{s-1})
-----------------------------
Construct M = M[0]M[1]…M[N-1]
A ← 0x67452301
B ← 0xEFCDAB89
C ← 0x98BADCFE
D ← 0x10325476
for i = 0 to N/16 do
   for j = 0 to 15 do X[j] = M[i·16+j]
   A‘ ← A
   B‘ ← B
   C‘ ← C
   D‘ ← D
   Round 1
   Round 2
   Round 3
   Round 4
   A ← A + A‘
   B ← B + B‘
   C ← C + C‘
   D ← D + D‘
-----------------------------
(h(m) = A || B || C || D)
```

Iteration of the compression function (for-loop body)

Cryptographic Hash Functions

Exemplary Hash Functions – MD5

Round 1

1. A ← (A + f(B,C,D) + X[0] + T[1]) ↲ 7
2. D ← (D + f(A,B,C) + X[1] + T[2]) ↲ 12
3. C ← (C + f(D,A,B) + X[2] + T[3]) ↲ 17
4. B ← (B + f(C,D,A) + X[3] + T[4]) ↲ 22
5. A ← (A + f(B,C,D) + X[4] + T[5]) ↲ 7
6. D ← (D + f(A,B,C) + X[5] + T[6]) ↲ 12
7. C ← (C + f(D,A,B) + X[6] + T[7]) ↲ 17
8. B ← (B + f(C,D,A) + X[7] + T[8]) ↲ 22
9. A ← (A + f(B,C,D) + X[8] + T[9]) ↲ 7
10. D ← (D + f(A,B,C) + X[9] + T[10]) ↲ 12
11. C ← (C + f(D,A,B) + X[10] + T[11]) ↲ 17
12. B ← (B + f(C,D,A) + X[11] + T[12]) ↲ 22
13. A ← (A + f(B,C,D) + X[12] + T[13]) ↲ 7
14. D ← (D + f(A,B,C) + X[13] + T[14]) ↲ 12
15. C ← (C + f(D,A,B) + X[14] + T[15]) ↲ 17
16. B ← (B + f(C,D,A) + X[15] + T[16]) ↲ 22

Round 2

1. A ← (A + g(B,C,D) + X[1] + T[17]) ↲ 5
2. D ← (D + g(A,B,C) + X[6] + T[18]) ↲ 9
3. C ← (C + g(D,A,B) + X[11] + T[19]) ↲ 14
4. B ← (B + g(C,D,A) + X[0] + T[20]) ↲ 20
5. A ← (A + g(B,C,D) + X[5] + T[21]) ↲ 5
6. D ← (D + g(A,B,C) + X[10] + T[22]) ↲ 9
7. C ← (C + g(D,A,B) + X[15] + T[23]) ↲ 14
8. B ← (B + g(C,D,A) + X[4] + T[24]) ↲ 20
9. A ← (A + g(B,C,D) + X[9] + T[25]) ↲ 5
10. D ← (D + g(A,B,C) + X[14] + T[26]) ↲ 9
11. C ← (C + g(D,A,B) + X[3] + T[27]) ↲ 14
12. B ← (B + g(C,D,A) + X[8] + T[28]) ↲ 20
13. A ← (A + g(B,C,D) + X[13] + T[29]) ↲ 5
14. D ← (D + g(A,B,C) + X[2] + T[30]) ↲ 9
15. C ← (C + g(D,A,B) + X[7] + T[31]) ↲ 14
16. B ← (B + g(C,D,A) + X[12] + T[32]) ↲ 20

Cryptographic Hash Functions

Exemplary Hash Functions – MD5

Round 3

1. $A \leftarrow (A + h(B,C,D) + X[5] + T[33]) \hookleftarrow 4$
2. $D \leftarrow (D + h(A,B,C) + X[8] + T[34]) \hookleftarrow 11$
3. $C \leftarrow (C + h(D,A,B) + X[11] + T[35]) \hookleftarrow 16$
4. $B \leftarrow (B + h(C,D,A) + X[14] + T[36]) \hookleftarrow 23$
5. $A \leftarrow (A + h(B,C,D) + X[1] + T[37]) \hookleftarrow 4$
6. $D \leftarrow (D + h(A,B,C) + X[4] + T[38]) \hookleftarrow 11$
7. $C \leftarrow (C + h(D,A,B) + X[7] + T[39]) \hookleftarrow 16$
8. $B \leftarrow (B + h(C,D,A) + X[10] + T[40]) \hookleftarrow 23$
9. $A \leftarrow (A + h(B,C,D) + X[13] + T[41]) \hookleftarrow 4$
10. $D \leftarrow (D + h(A,B,C) + X[0] + T[42]) \hookleftarrow 11$
11. $C \leftarrow (C + h(D,A,B) + X[3] + T[43]) \hookleftarrow 16$
12. $B \leftarrow (B + h(C,D,A) + X[6] + T[44]) \hookleftarrow 23$
13. $A \leftarrow (A + h(B,C,D) + X[9] + T[45]) \hookleftarrow 4$
14. $D \leftarrow (D + h(A,B,C) + X[12] + T[46]) \hookleftarrow 11$
15. $C \leftarrow (C + h(D,A,B) + X[15] + T[47]) \hookleftarrow 16$
16. $B \leftarrow (B + h(C,D,A) + X[2] + T[48]) \hookleftarrow 23$

Round 4

1. $A \leftarrow (A + i(B,C,D) + X[0] + T[49]) \hookleftarrow 6$
2. $D \leftarrow (D + i(A,B,C) + X[7] + T[50]) \hookleftarrow 10$
3. $C \leftarrow (C + i(D,A,B) + X[14] + T[51]) \hookleftarrow 15$
4. $B \leftarrow (B + i(C,D,A) + X[5] + T[52]) \hookleftarrow 21$
5. $A \leftarrow (A + i(B,C,D) + X[12] + T[53]) \hookleftarrow 6$
6. $D \leftarrow (D + i(A,B,C) + X[3] + T[54]) \hookleftarrow 10$
7. $C \leftarrow (C + i(D,A,B) + X[10] + T[55]) \hookleftarrow 15$
8. $B \leftarrow (B + i(C,D,A) + X[1] + T[56]) \hookleftarrow 21$
9. $A \leftarrow (A + i(B,C,D) + X[8] + T[57]) \hookleftarrow 6$
10. $D \leftarrow (D + i(A,B,C) + X[15] + T[58]) \hookleftarrow 10$
11. $C \leftarrow (C + i(D,A,B) + X[6] + T[59]) \hookleftarrow 15$
12. $B \leftarrow (B + i(C,D,A) + X[13] + T[60]) \hookleftarrow 21$
13. $A \leftarrow (A + i(B,C,D) + X[4] + T[61]) \hookleftarrow 6$
14. $D \leftarrow (D + i(A,B,C) + X[11] + T[62]) \hookleftarrow 10$
15. $C \leftarrow (C + i(D,A,B) + X[2] + T[63]) \hookleftarrow 15$
16. $B \leftarrow (B + i(C,D,A) + X[9] + T[64]) \hookleftarrow 21$

Cryptographic Hash Functions

Exemplary Hash Functions – MD5

- Until 2004, MD4 was considered to be insecure and MD5 was considered to be partially broken
  - In 1993, Bert den Boer and Antoon Bosselaers found collisions for the compression function of MD5 (i.e., they found pairs of different message blocks and chaining values that compress to the same value
  - In 1996, Hans Dobbertin found collisions for different message blocks that employ the same chaining value different from the true IV employed by MD5)

  - In 2004, Xiaoyun Wang et al. (Shandong University, China) found collisions for MD4, MD5, and many other cryptographic hash functions
  - Consequently, MD5 is broken today and should no longer be used

Cryptographic Hash Functions

Exemplary Hash Functions – SHA-1

- SHA-1 is a cryptographic hash function developed and proposed by the **National Institute of Standards and Technology**
- It is conceptually and structurally similar to MD4 and MD5 (Merkle-Damgård construction with b = 512 and l = 160)
- Major differences
  - SHA-1 is designed and optimized for computer systems with a big-endian architecture
  - SHA-1 employs 5 registers (instead of 4)
  - SHA-1 hash values are 5·32 = 160 bits long
- The preprocessing of message m and array M is the same as with MD4 and MD5 (but for a big-endian architecture)

Cryptographic Hash Functions

Exemplary Hash Functions – SHA-1

- SHA-1 employs a sequence of 80 Boolean functions $f_0,f_1,\ldots,f_{79}$

$$f_t(X,Y,Z) = \begin{cases} 0 \le t \le 19: \mathrm{Ch}(X,Y,Z) = (X \wedge Y) \oplus ((\neg X) \wedge Z) \\ 20 \le t \le 39: \mathrm{Parity}(X,Y,Z) = X \oplus Y \oplus Z \\ 40 \le t \le 59: \mathrm{Maj}(X,Y,Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) \\ 60 \le t \le 79: \mathrm{Parity}\ (X,Y,Z) = X \oplus Y \oplus Z \end{cases}$$

- SHA-1 employs a sequence of 80 constant 32-bit words $K_0,K_1,\ldots,K_{79}$

$$K_t = \begin{cases} \lfloor 2^{30}2^{1/2} \rfloor = \texttt{0x5A827999} & 0 \le t \le 19 \\ \lfloor 2^{30}3^{1/2} \rfloor = \texttt{0x6ED9EBA1} & 20 \le t \le 39 \\ \lfloor 2^{30}5^{1/2} \rfloor = \texttt{0x8F1BBCDC} & 40 \le t \le 59 \\ \lfloor 2^{30}10^{1/2} \rfloor = \texttt{0xCA62C1D6} & 60 \le t \le 79 \end{cases}$$

Cryptographic Hash Functions

Exemplary Hash Functions – SHA-1

- Furthermore, SHA-1 also employs a message schedule W
- The respective algorithm takes as input 16 32-bit words (i.e., 512 bits) from M and generates as ouput 80 32-bit words (i.e., 2,560 bits)

$$W_t = \begin{cases} M[t] & 0 \leq t \leq 19 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 20 \leq t \leq 79 \end{cases}$$

Cryptographic Hash Functions

Exemplary Hash Functions – SHA-1

Each M[i] (i = 0,...,N-1) consists of 16 32-bit words

- SHA-1 algorithm (overview)

```
(m = $m_0 m_1 \ldots m_{s-1}$)
Construct M = M[0]M[1]…M[N-1]
A ← 0x67452301; B ← 0xEFCDAB89
C ← 0x98BADCFE; D ← 0x10325476
E ← 0xC3D2E1F0
for i = 0 to N-1 do
   Expand message schedule W from M[i]
   A‘ ← A; B‘ ← B; C‘ ← C; D‘ ← D; E‘ ← E
   for t = 0 to 79 do
      $T \leftarrow (A \hookleftarrow 5) + f_t(B,C,D) + E + K_t + W_t$
      E ← D
      D ← C
      $C \leftarrow B \hookleftarrow 30$
      B ← A
      A ← T
   A ← A + A‘
   B ← B + B‘
   C ← C + C‘
   D ← D + D‘
   E ← E + E‘
(h(m) = A || B || C || D || E)
```

Cryptographic Hash Functions

Exemplary Hash Functions – SHA-1

- SHA-1 is conceptually and structurally similar to MD4 and MD5, and hence it appears to be vulnerable to the same attacks
- But a SHA-1 hash value is 32 bit longer than an MD5 hash value → it is potentially more collision resistant than MD5
- In 2005, Wang et al. found an attack against the collision resistance property of SHA-1 that requires 2^{69} hash operations (instead of 2^{80})
- The attack has been improved to 2^{63}
- People therefore move away from SHA-1 to SHA-2 and other crypto-graphic hash functions (if possible)

Cryptographic Hash Functions

9.4 Final Remarks

- Most cryptographic hash functions in use today follow the Merkle-Damgård construction
- Finding collisions for such functions has become an active area of research
- Furthermore, the block chaining structure of iterative hash functions prevents parallelism → performance bottleneck
- It is a research challenge to design cryptographic hash functions that are inherently (and provably) more secure and qualified to support parallelism
- The NIST runs a competition for one (or several) standardized cryptographic hash function(s) until 2012
  → http://csrc.nist.gov/groups/ST/hash/sha-3/

10 Digital Signature Systems

Digital Signature Systems

10.1 Introduction

- A DSS with appendix consists of 3 efficiently computable algorithms
  - Generate(1^l)
  - Sign(k^{-1},m)
  - Verify(k,m,s)
- A DSS giving message recovery consists of 3 efficiently computable algorithms
  - Generate(1^l)
  - Sign(k^{-1},m)
  - Recover(k,s)

Digital Signature Systems

Introduction

- Basic requirements for a DSS
  - **Correctness** → A valid signature must be accepted
  - **Security** → It must be impossible or computationally infeasible to forge (i.e., illegitimately generate) a valid signature
- There are different interpretations for the security requirement
- As usual, a precise security definition must specify
  - The adversary (including his or her capabilities)
  - The task he or she must solve to be successful, i.e., break the security of the system
- The terminology most frequently used was proposed by Shafi Goldwasser, Silvio Micali, and Ron Rivest in the 1980s

Digital Signature Systems

Introduction

- Adversary
  - Computing power is polynomially bounded (with respect to the length of the input string)
  - Possible attacks
    - Key-only attack
    - Known message attack
    - Chosen message attack
      - Generic chosen message attack
        → attack strategy does not depend on the public key
      - Directed chosen message attack
        → attack strategy depends on the public key
      - Adaptive chosen message attack

Digital Signature Systems

Introduction

- Task to solve
  - Total break
  - Universal forgery
  - Selective forgery
  - Existential forgery
- A DSS is **provably secure** if it can be shown that a polynomially bounded adversary who can mount adaptive chosen message attacks is not even able to existentially forge a signature
- Many deployed DSSs are not provably secure

Digital Signature Systems

10.2 Basic Systems

- RSA
- Elgamal
- Schnorr
- DSA

Digital Signature Systems

Basic Systems – RSA

- The RSA public key cryptosystem also yields a DSS
- The RSA key Generation Algorithm **Generate(1^l)** is the same as for the RSA asymmetric encryption system
- It is probabilistic and outputs a public key pair $(k,k^{-1}) = ((n,e),d)$
- Toy example
  - $p = 11$ and $q = 23 \rightarrow n = 253$ and $\phi(n) = 10 \cdot 22 = 220$
  - Public verification key is $(n,e) = (253,3)$
  - Private signing key is $d = 147$

Digital Signature Systems

Basic Systems – RSA

- The RSA Signature Generation Algorithm **Sign(k^{-1},m)** is deterministic
- It takes as input a private signing key $k^{-1} = d$ and a message $m \in \mathbf{Z}_n$, and it generates as output a digital signature s
  - If RSA is used as DSS giving message recovery, then m must be sufficiently small (i.e., $m < n$)
  - If RSA is used as a DSS with appendix, then m must either be sufficiently small or hashed to a bit string of fixed size

$$s = RSA_{n,d}(m) \equiv m^d \pmod{n} \text{ or } h(m)^d \pmod{n}$$

Digital Signature Systems

Basic Systems – RSA

- In either case, it is necessary to expand m or h(m) to the size of the RSA modulus n
- This can be done by prepending zeros or using a message expansion function (preferred choice)
  - In theory, there are many message expansion functions to choose from
  - In practice, there are only a few functions in widespread use
- Deterministic message expansion functions
  - ANSI X9.31
    - o $h_{ANSI\ X9.31}(m)$ = 6B BB BB … BB BA || h(m) || 3x CC
  - PKCS #1
    - o $h_{PKCS\ \#1}(m)$ = $\underleftrightarrow{\text{00 01 FF … FF}}$ 00 || h_{ID} || h(m)

x = 1: RIPEMD-160

x = 3: SHA-1

Identifier for the cryptographic hash function in use

There are so many FF bytes (representing 11111111 in binary notation or 255 in decimal notation) that the total bit length of $h_{PKCS\#1}(m)$ equals the bit length of n

Digital Signature Systems

Basic Systems – RSA

- Alternatively, there are probabilitic message expansion functions
- Most importantly, the **probabilistic signature scheme (PSS)** uses random values to expand h(m) to the bit length of n
- RSA-PSS is provably secure in the random oracle model

Digital Signature Systems

Basic Systems – RSA

- If RSA is used as a DSS giving message recovery, then it is sufficient to transmit s (i.e., m need not be transmitted)
- If RSA is used as a DSS with appendix, then s must be transmitted along with the message m
- Toy example (m or $h(m) = 26 \in \mathbf{Z}_{253}$)
  - $s \equiv m^d \pmod{n} \equiv 26^{147} \pmod{253} = 104$
- For the purpose of signature verification, one must distinguish whether the RSA DSS is used with appendix or giving message recovery
- In either case, the corresponding RSA Signature Verification Algorithm (i.e., **Verify** or **Recover**) is deterministic and efficient
- It requires one modular exponentiation and optionally the invocation of a cryptographic hash function

Digital Signature Systems

Basic Systems – RSA

- If the RSA DSS is used with appendix, then **Verify(k,m,s)** takes as input a public verification key k = (n,e), a message m, and a signa-ture s, and it generatesas output one bit indicating whether s is a valid signature for m with respect to k
- The algorithm comprises 2 steps
  - It computes $m' = RSA_{n,e}(s) \equiv s^e \pmod{n}$
  - It compares m' with m or h(m)
- Toy example
  - $m' = RSA_{253,3}(104) \equiv 104^3 \pmod{253} = 26$
  - Signature is valid (m = m')

Digital Signature Systems

Basic Systems – RSA

- If the RSA DSS is used giving message recovery, then **Recover(k,s)** takes as input a public verification key k = (n,e), and a signature s, and it outputs the message m or a notification indicating that s is not a valid signature
- The algorithm comprises 2 steps
  - It computes $m = RSA_{n,e}(s) \equiv s^e \pmod{n}$
  - It decides whether m is a valid message

If yes, then the algorithm returns m

If no, then the algorithm returns a notification indicating that s is not a valid signature for m with respect to k

- The second step is important
- If every message represented a valid message, then an adversary could trivially find a valid (i.e., existentially forged) RSA signature s by randomly selecting $s \in \mathbf{Z}_n$ and claiming that it is a valid signature for $m \equiv s^e \pmod{n}$

Digital Signature Systems

Basic Systems – RSA

- In the toy example, for example, the RSA Recover algorithm must compute

$$m = RSA_{253,3}(104) \equiv 104^3 \pmod{253} = 26$$

  and decide whether m = 26 is a valid message
- The decision depends on the system in use
- If all valid messages must be congruent to 6 modulo 20, then m = 26 is a valid message

Digital Signature Systems

Basic Systems – RSA

- Most security properties of the RSA asymmetric encryption system also apply for the RSA DSS
- The fact that the RSA function is multiplicatively homomorphic is particularly dangerous
- If m_1 and m_2 are two messages with signatures s_1 and s_2, then

  $$s = s_1 s_2 \equiv m_1^d m_2^d \equiv (m_1 m_2)^d \pmod{n}$$

  is also a valid signature for $m \equiv m_1 m_2 \pmod{n}$
- Best practices in security engineering must take care of this fact and protect against corresponding attacks
- One can either require that messages have a certain (non-multiplicative) structure or randomly pad the messages prior to signing

Digital Signature Systems

Basic Systems – RSA

- In many situations, RSA is used to encrypt and digitally sign a message
- Hence, it may be necessary to appliy both the RSA **Encrypt** and the RSA **Sign** algorithms
- The question is whether the order of the operations matters (i.e., Encrypt-then-sign or Sign-then-Encrypt)
- In the general case, the answer is not clear and it depends on the purpose of the cryptographic protection
- In many situations, however, Sign-then-encrypt is preferred (so the signer sees the message he or she is about to sign in the clear)
- In either case, one must be cautious about the relative sizes of the moduli → the message may need to be reblocked (reblocking problem)

Digital Signature Systems

Basic Systems – RSA

- In summary, the RSA DSS is considered to be secure
- This is particularly true if the modulus n is sufficiently large (≥ 1,024 bits)
- Because digital signatures are used to protect valuable data for potentially long periods of time, it is often recommended to use longer moduli, such as 2,048 or 4,096 bits
- It is also recommended to use RSA as a DSS with appendix with appropriately chosen hash and message expansion functions
  - SHA-1
  - PKCS #1 or PSS

Digital Signature Systems

Basic Systems – Elgamal

- The Elgamal public key cryptosystem yields a DSS with appendix
- There are variations of Elgamal that yield DSSs giving message recovery (e.g., Nyberg-Rueppel)
- The Elgamal DSS is not as widely deployed as the RSA DSS
  - Elgamal employs different algorithms for encryption / decryption and signature generation / verification
  - Elgamal signatures are twice as long as RSA signatures
- Again, the security of Elgamal is based on the DLP in a cyclic group (e.g., $\mathbf{Z}_p^*$)
- Any cyclic group (in which the DLP is intractable) can be used to instatiate the Elgamal DSS (e.g., group of points on an elliptic curve over a finite field)

Digital Signature Systems

Basic Systems – Elgamal

- The Elgamal Key Generation Algorithm **Generate(1^l)** is the same as for the Elgamal asymmetric encryption system
- The public (signature verification) key is (p,g,y), where p and g may be system parameters
- The private (signing) key is x with $y \equiv g^x \pmod{p}$
- Toy example
  - $p = 17$ and $g = 7$
  - $x = 6$
  - $y \equiv 7^6 \equiv 117{,}649 \pmod{17} = 9$

Digital Signature Systems

Basic Systems – Elgamal

- The Elgamal Signature Generation Algorithm **Sign(k^{-1},m)** is probablistic and employs a cryptographic hash function h

Note that r must be fresh and unique for every message that is signed

Also note that it must be kept secret and not leak the implementation – otherwise the private key x may get compromised

$$\overbrace{(p,g,x,m)}^{k^{-1}}$$

$r \in_R \mathbf{Z}_p^*$

$s_1 \equiv g^r \pmod p$

$s_2 \equiv (r^{-1}(h(m)-xs_1)) \pmod{(p-1)}$

(s_1,s_2)

- Toy example
  - m with (h(m) = 6 and r = 3
  - $s_1 \equiv g^r \pmod p \equiv 7^3 \pmod{17} = 3$
  - $r^{-1} \pmod{p-1} \equiv 3^{-1} \pmod{16} = 11$
  - $s_2 \equiv (r^{-1}(h(m)-xs_1))(\bmod\ p-1)$
    $\equiv (11(6 - 6 \cdot 3))(\bmod\ 16)$
    $\equiv (11(6 - 18))(\bmod\ 16)$
    $\equiv (11(-12))(\bmod\ 16)$
    $\equiv (-132)(\bmod\ 16)$
    $\equiv (-4)(\bmod\ 16) = 12$
  - The Elgamal signature is (3,12)

Digital Signature Systems

Basic Systems – Elgamal

- The Elgamal Sign algorithm can be optimized using precomputation
- The signatory can select $r \in_R \mathbf{Z}_p^*$ and precompute
  - $s_1 \equiv g^r \pmod p$
  - $r^{-1} \pmod{p-1}$
- It can then digitally sign message m by computing

$$s_2 \equiv (r^{-1}(h(m)-xs_1))\pmod{p-1}$$

- The Elgamal signature for message m is (s_1,s_2) with $s_1,s_2 \in \mathbf{Z}_p^*$ (twice as long as an RSA signature)

Digital Signature Systems

Basic Systems – Elgamal

- The Elgamal Signature Verification Algorithm **Verify(k,m,s)** is deterministic
- It must verify the relation

$$1 \le s_1 \le p-1$$

and the equivalence

$$g^{h(m)} \equiv y^{s_1} s_1^{s_2} \pmod{p}$$

$$\begin{aligned} y^{s_1} s_1^{s_2} &\equiv g^{xs_1} g^{rr^{-1}(h(m)-xs_1)} \pmod{p} \\ &\equiv g^{xs_1} g^{(h(m)-xs_1)} \pmod{p} \\ &\equiv g^{xs_1} g^{-xs_1} g^{h(m)} \pmod{p} \\ &\equiv g^{h(m)} \pmod{p} \end{aligned}$$

- The signature is valid iff both checks are positive (without the first check, it is possible to forge a new Elgamal signature from a given one)
- Toy example

$$1 \le 3 \le 16 \text{ and } 7^6 \pmod{17} \equiv 117{,}649 \pmod{17} = 9$$

$$9^3 3^{12} \pmod{17} \equiv 387{,}420{,}489 \pmod{17} = 9$$

Digital Signature Systems

Basic Systems – Elgamal

- The security of the Elgamal DSS depends on the DLA and the assumed intractability of the DLP in the cyclic group in use
- If $\mathbf{Z}_p^*$ is used, then
  - p should be at least 1,024 bits long
  - p-1 should not have only small prime factors (otherwise the Pohlig-Hellman algorithm can be used to efficiently solve the DLP)

Digital Signature Systems

Basic Systems – Schnorr

- In the late 1980s, Claus-Peter Schnorr proposed a modification of the Elgamal DSS
- The basic idea is to do the computations in a subgroup of $\mathbf{Z}_p^*$ (instead of $\mathbf{Z}_p^*$)
- The resulting Schnorr signatures are shorter and the computations can be done more efficiently

Digital Signature Systems

Basic Systems – Schnorr

- The Schnorr Key Generation Algorithm **Generate(1^l)** is probabilistic
  - It randomly selects 2 large primes p and q with $q \mid p-1$ (typically, $|p|=1024$ and $|q|=160$)
  - It selects a generator g of a q-element subgroup of $\mathbf{Z}_p^*$
  - It randomly selects a private (signing) key $0 < x < q$
  - It computes a corresponding public (signature verification) key $y \equiv g^x \pmod p$
- Toy example
  - $p = 23$, $\mathbf{Z}_{23}^* = \{1,2,\ldots,22\}$ with $|\mathbf{Z}_{23}^*| = 22$
  - $q = 11$ (note that $11 \mid 23-1$)
  - $g = 2$ ($\langle 2 \rangle = \{1,2,3,4,6,8,9,12,13,16,18\}$ and $|\langle 2 \rangle| = 11$)
  - $x = 5$ and $y \equiv 2^5 \equiv 32 \pmod{23} = 9$

Digital Signature Systems

Basic Systems – Schnorr

- The Schnorr Signature Generation Algorithm **Sign(k^{-1},m)** is probabilistic and employs a cryptographic hash function h

$$\overbrace{(p,q,g,x}^{k^{-1}},m)$$

$r \in_R \mathbf{Z}_p^*$
$s \equiv g^r \pmod{p}$
$s_1 = h(m||s)$
$s_2 \equiv xs_1 + r \pmod{q}$

(s_1,s_2)

- Toy example
  - $r = 7$
  - $s \equiv 2^7 \equiv 128 \pmod{23} = 13$
  - $s_1 = h(m||13) = 4$ (assumption)
  - $s_2 \equiv 5 \cdot 4 + 7 \equiv 27 \pmod{11} = 5$
  - The Schnorr signature is $(s_1,s_2) = (4,5)$

Digital Signature Systems

Basic Systems – Schnorr

- Schnorr Signature Verification Algorithm
  - $u \equiv g^{s_2} y^{-s_1} \pmod{p}$
  - $t = h(m||u)$
  - Signature (s_1,s_2) is valid $\Leftrightarrow t = s_1$
  - Note that $u \equiv g^{s_2} y^{-s_1} \equiv g^{s_2} y^{-xs_1} \equiv g^r \equiv s \pmod{p}$ and $t = h(m||u) = h(m||s) = s_1$
- Toy example
  - $u \equiv 2^5 9^{-4} \equiv 2^5 9^7 \equiv 32 \cdot 4‘782‘969 \equiv 153‘055‘008 \pmod{23} = 13$
  - $t = h(m||13) = 4 = s_1$

Digital Signature Systems

Basic Systems – Schnorr

- The Schnorr DSS is a modified version of Elgamal
- Its security is comparable
- Unlike Elgamal, the Schnorr DSS relies on the DLP in a subgroup of $\mathbf{Z}_p^*$ with prime order q
- Solution requires a generic algorithm with a running time that is of the order of the square root of the order of the subgroup
- If the subgroup has order 2^{160}, then the best known algorithm to compute discrete logaritms has a running time of order $2^{160/2} = 2^{80}$

Digital Signature Systems

Basic Systems – DSA

- Based on the Elgamal and Schnorr DSSs, the U.S. NIST developed and proposed the **Digital Signature Algorithm (DSA)**
- In 1994, the correspondig **Digital Signature Standard (DSS)** was specified in FIPS PUB 186 (revised three times)
- The DSA employs SHA-1
- The acronym ECDSA refers to the elliptic curve version of the DSA (standardized by ANSI X9F1, IEEE P1363, …)

Digital Signature Systems

Basic Systems – DSA

- DSA Key Generation Algorithm **Generate(1^l)** operates in 2 steps
  - It determines 2 appropriately sized prime modulo p and q
    - p must be 512+64t bits long ($t \in \{0,\ldots,8\}$)
    - q must be 160 bits long, i.e., $2^{159} < q < 2^{160}$, and divide p-1

    $q \mid p-1 \Rightarrow \mathbf{Z}_p^*$ has a subgroup of order q

    The subgroup can be found by using $h \in (1,p-1)$ with $h^{(p-1)/q} \pmod{p} > 1$ and computing the generator

    $$g \equiv h^{(p-1)/q} \pmod{p}$$

  - For every user, it randomly selects a private (signing) key $x \in \mathbf{Z}_q$ and computes a public (signature verification) key $y \equiv g^x \pmod{p}$
- p, q, and g may be system parameters
- Otherwise, they must be part of the public key

Digital Signature Systems

Basic Systems – DSA

- Toy example
  - $p = 23$
  - $q = 11$ (note that $11 \mid 22$)
  - For $h = 2$, $g \equiv 2^{22/11} \equiv 2^2 \pmod{23} = 4$
  - Private (signing) key $x = 3$
  - Public (signature verification) key $y \equiv 4^3 \pmod{23} = 18$

Digital Signature Systems

Basic Systems – DSA

- The DSA Signature Generation Algorithm **Sign(k^{-1},m)** is probablistic

$$\overbrace{(p,q,g,x,m)}^{k^{-1}}$$

$r \in_R \mathbf{Z}_q^*$

$s_1 \equiv (g^r \pmod p) \pmod q$

$s_2 \equiv (r^{-1}(h(m)+xs_1)) \pmod q$

(s_1,s_2)

s_1 and s_2 are 160-bit numbers $\Rightarrow$
A DSA signature is 320 bits long

- Toy example (m with h(m) = 6)
  - $r = 7$
  - $r^{-1} \equiv 7^{-1} \pmod{11} = 8$
  - $s_1 \equiv (4^7 \pmod{23}) \pmod{11} \equiv 8 \pmod{11} = 8$
  - $s_2 \equiv (8(6+3 \cdot 8)) \pmod{11} \equiv 8 \cdot 30 \pmod{11} = 9$
  - Hence, $(s_1,s_2) = (8,9)$ represents a DSA signature for h(m) = 6

Digital Signature Systems

Basic Systems – DSA

- The DSA Signature Verification Algorithm **Verify(k,m,s)** must first verify that

  $$0 < s_1,s_2 < q$$

  and then compute

  $$w \equiv s_2^{-1} \pmod q$$
  $$u_1 \equiv h(m)w \pmod q$$
  $$u_2 \equiv s_1 w \pmod q$$
  $$v \equiv (g^{u_1} y^{u_2} \pmod p) \pmod q$$

- The signature is valid $\Leftrightarrow v = s_1$

- Toy example (m with h(m) = 6)
  - $1 \leq 8,9 \leq 10$
  - $w \equiv 9^{-1} \pmod{11} = 5$
  - $u_1 \equiv 6 \cdot 5 \pmod{11} \equiv 30 \pmod{11} = 8$
  - $u_2 \equiv 8 \cdot 5 \pmod{11} \equiv 40 \pmod{11} = 7$
  - $v \equiv (4^8 18^7 \pmod{23}) \pmod{11} \equiv (65{,}536 \cdot 612{,}220{,}032 \pmod{23}) \pmod{11} \equiv (40{,}122{,}452{,}017{,}152 \pmod{23}) \pmod{11} \equiv 8 \pmod{11} = 8$
  - Signature is valid (v = s1 = 8)

Digital Signature Systems

10.3 Secure Systems

- The notion of a (provably) secure DSS was introduced by Goldwasser, Micali, and Rivest in the 1980s
- Their construction is not efficient
- This is also partly true for a construction proposed by Cynthia Dwork and Moni Naor

- Towards the end of the 1990s, a few cryptographic researchers proposed DSSs that are provably secure and efficient
  - Bellare-Rogaway (1996)
    - Probabilistic signature scheme (PSS)
    - PSS with recovery (PSS-R)

    PSS and PSS-R are provably secure in the random oracle model
  - Cramer-Shoup (1999)
  - Gennaro-Halevi-Rabin (1999)

Digital Signature Systems

Secure Systems – PSS

- PSS represents a DSS with appendix
- The basic idea is „random padding“, i.e., replace a deterministic message expansion function with a probabilistic one
- The expanded message is then digitally signed with a conventional DSS (e.g., PSS-RSA)
- In addition to $k = \log n$, the PSS requires 2 additional parameters $0 < k_0, k_1 < k$ (typically $k_0 = k_1 = 128$)
- The PSS employs 2 cryptographic hash functions
  - Compressor $h : \{0,1\}^* \rightarrow \{0,1\}^{k_1}$
  - Generator $g : \{0,1\}^{k_1} \rightarrow \{0,1\}^{k-k_1-1}$
    - g_1 is a function that on input $w \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(w)$
    - g_2 is a function that on input $w \in \{0,1\}^{k_1}$ returns the remaining $k-k_0-k_1-1$ bits of $g(w)$

Digital Signature Systems

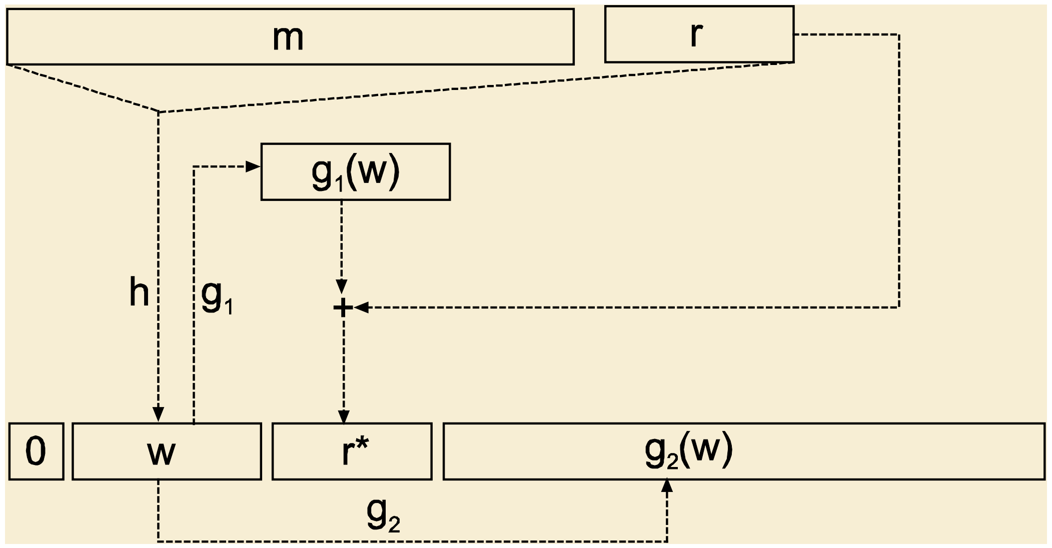
Secure Systems – PSS

- As its name suggests, the PSS-RSA Signature Generation Algorithm **Sign(k^{-1},m)** is probabilistic

$$\overbrace{(n,d}^{k^{-1}},m)$$

$$
\begin{aligned}
& r \in_R \{0,1\}^{k_0} \\
& \quad w \leftarrow h(m) \,||\, r \\
& \quad r^* \leftarrow g_1(w) \oplus r \\
& \quad y \leftarrow 0 \,||\, w \,||\, r^* \,||\, g_2(w) \\
& \quad s \leftarrow y^d \pmod{n}
\end{aligned}
$$

(s)



Digital Signature Systems

Secure Systems – PSS

- The PSS-RSA Signature Verification Algorithm **Verify(k,m,s)** is deterministic

This is possible, because every component has a fixed length

$$(\underbrace{n,e}_{k},m,s)$$

```
$y \leftarrow s^e \pmod{n}$
break up $y$ as $b \,||\, w \,||\, r^* \,||\, \gamma$
$r \leftarrow r^* \oplus g_1(w)$
$B \leftarrow (b = 0 \text{ and } h(m||r) = w \text{ and } g_2(w) = \gamma)$
```

(B)

B is a Boolean predicate

Digital Signature Systems

Secure Systems – PSS-R

- PSS-R represents a DSS giving message recovery
- The functions h, g, g_1, and g_2 are identically defined
- If the message is sufficiently short, then one can fold the entire message into the signature
- More specifically, the messages to be signed have length $k_m = k - k_0 - k_1 - 1$
- Suggested choices
  - $k = 1{,}024$
  - $k_0 = k_1 = 128$
  - $k_m = 767$ bits

Digital Signature Systems

Secure Systems – PSS-R

- Again, the PSS-R-RSA Signature Generation Algorithm **Sign(k^{-1},m)** is probabilistic

$$(\overbrace{n,d}^{k^{-1}},m)$$

$r \in_R \{0,1\}^{k_0}$

$w \leftarrow h(m) \| r$

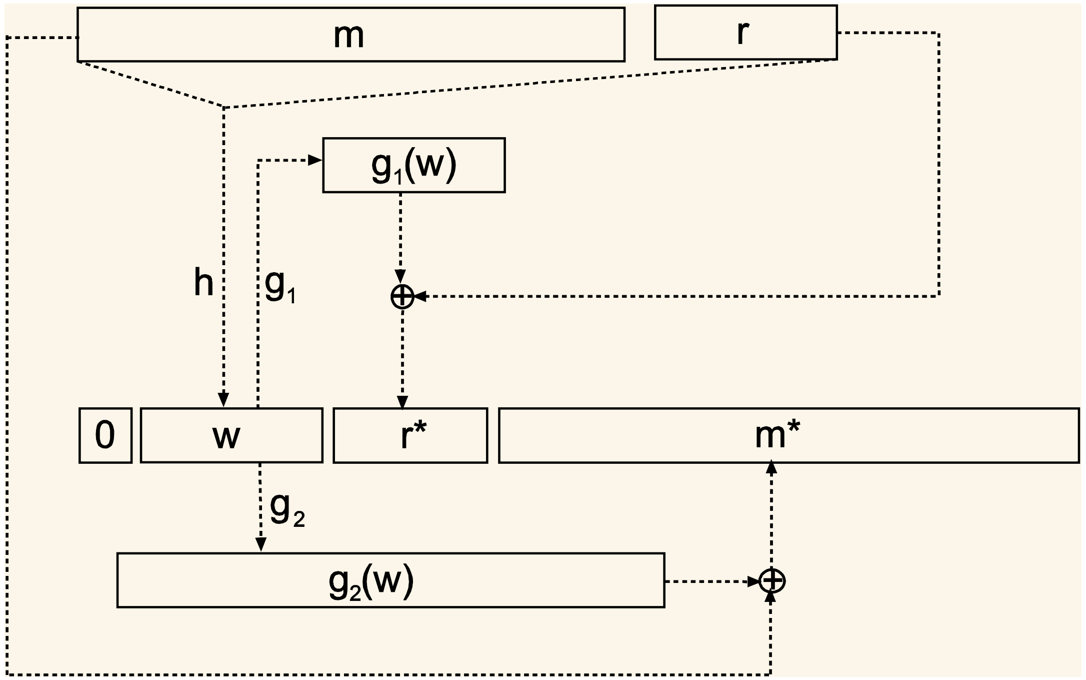
$r^* \leftarrow g_1(w) \oplus r$

$m^* \leftarrow g_2(w) \oplus m$

$y \leftarrow 0 \| w \| r^* \| m^*$

$s \leftarrow y^d \pmod n$

(s)



Digital Signature Systems

Secure Systems – PSS-R

- The PSS-R-RSA Recover Algorithm **Recover(k,s)** is deterministic

$\underbrace{(n,e}_{k},s)$

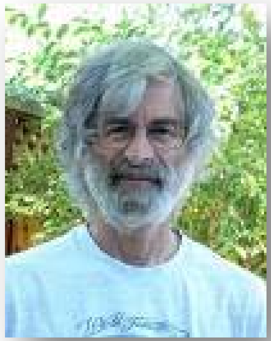
```
y ← $s^e \pmod n$
break up y as $b \| w \| r^* \| m^*$
$r \leftarrow r^* \oplus g_1(w)$
$m \leftarrow m^* \oplus g_2(w)$
If ($b = 0$ and $h(m\|r) = w$) then output m
                              else output *invalid*)
```

(m | *invalid*)

Digital Signature Systems

10.4 One-Time Signature Systems

- A one-time signature system is a DSS in which a new key pair is required for every message that is signed
  - The advantages are simplicity and efficiency
  - The disadvantages are related to the size of the verification key(s) and the complexity of key management
- When combined with techniques to efficiently authenticate verification keys, one-time signature systems are practical
- In 1978, Rabin proposed the idea and the first (inefficient) one-time signature system
- In 1979, Leslie Lamport proposed the first one-time signature system that is efficient and can actually be used in practice



Digital Signature Systems

One-Time Signature Systems

- The Lamport one-time signature system employs a one-way function f to digitally sign message m
- m is assumed to be at most n bits (e.g., n = 128 or 160 bits)
- If m is longer than n bits, then it must be hashed using a cryptographic hash function h (e.g., SHA-1)
- The message can then be written as $m = m_1m_2\ldots m_n$, where each m_i ($i = 1,\ldots,n$) represents a bit that is signed indivi-dually
- The signatory‘s private key comprises n pairs of randomly chosen preimages for f

$$[u_{10},u_{11}],[u_{20},u_{21}],\ldots,[u_{n0},u_{n1}]$$

- Each uij (i = 1,…,n; j = 0,1) may, for example, be a 64-bit string
- The 2n preimages can be generated with a PRBG (and a seed)

Digital Signature Systems

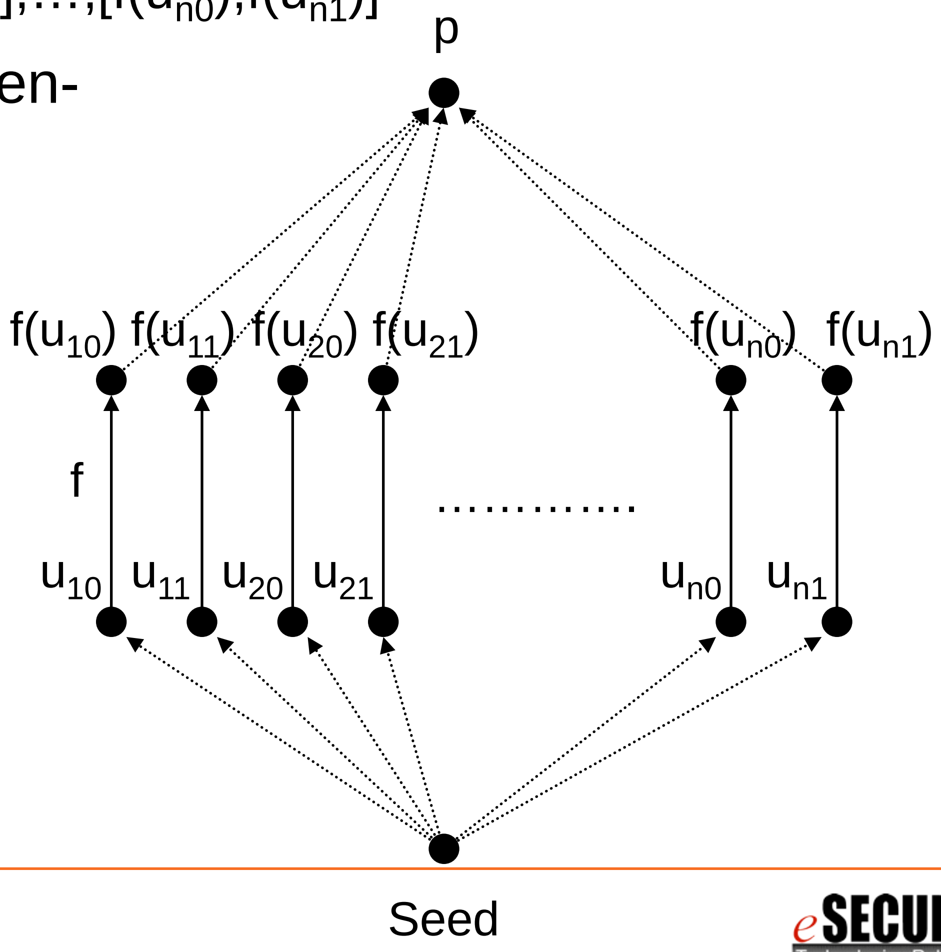
One-Time Signature Systems

- The public key comprises the images of the uij (i = 1,…,n; j = 0,1)

$$[f(u_{10}),f(u_{11})],[f(u_{20}),f(u_{21})],\ldots,[f(u_{n0}),f(u_{n1})]$$

- Again, in an efficient implementation, the 2n images can be hashed to a single value p

$$p = h([f(u_{10}),f(u_{11})],\\ [f(u_{20}),f(u_{21})],\\ \ldots,\\ [f(u_{n0}),f(u_{n1})]$$



Digital Signature Systems

One-Time Signature Systems

- To digitally sign message m, each message bit m_i ($i = 1,\dots,n$) is signed individually using $[u_{i_0},u_{i_1}]$
- More specifically, the signature for message bit m_i is the preimage u_{im_i}
  - If $m_i = 0$, then the bit is signed by u_{i0}
  - If $m_i = 1$, then the bit is signed by u_{i1}
- The resulting signature s consists of all u_{im_i} for $i = 1,\dots,n$
- s can be verified by computing all images $f(u_{ij})$, hashing all values to p‘, and comparing p‘ with p
- For example, the Lamport one-time signature for $m = 0110$ is $[u_{10},u_{21},u_{31},u_{40}]$
- There are many possibilities to generalize and improve the (efficiency of the) Lamport one-time signature system

Digital Signature Systems

10.5 Variations

- There are many variations of „normal“ DSSs
  - Blind signatures
  - Undeniable signatures
  - Group signatures
  - Fail-stop signatures
  - …

Digital Signature Systems

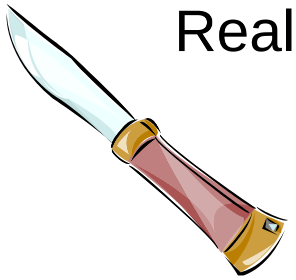
10.6 Final Remarks

- In addition to RSA, Elgamal, Schnorr, DAS, PSS, and PSS-R, there are many other DSSs proposed in the literature
- There are also DSSs derived from zero-knowledge authentication protocols
- It is hoped that
  - Digital signatures and DSSs provide the digital counterpart to hand-written signatures
  - They can provide nonrepudiation services
- Many countries have put forth legislation regarding digital signatures and their use in e-commerce
- Digital signature legislation has not been successful so far

Digital Signature Systems

Final Remarks

- Digital signature laws have not been challenged in the court, so their legal value remains unclear
- The fact that digital signatures are based on mathematical formulae intuitively makes us believe that the evidence they provide is strong
- This belief is seductive and sometimes wrong
- Digital signatures are digital objects, and as such they may be subject to multiple representations and interpretations



Real object

One representation (rendered by physics)

A few plausible interpretations

Digital object

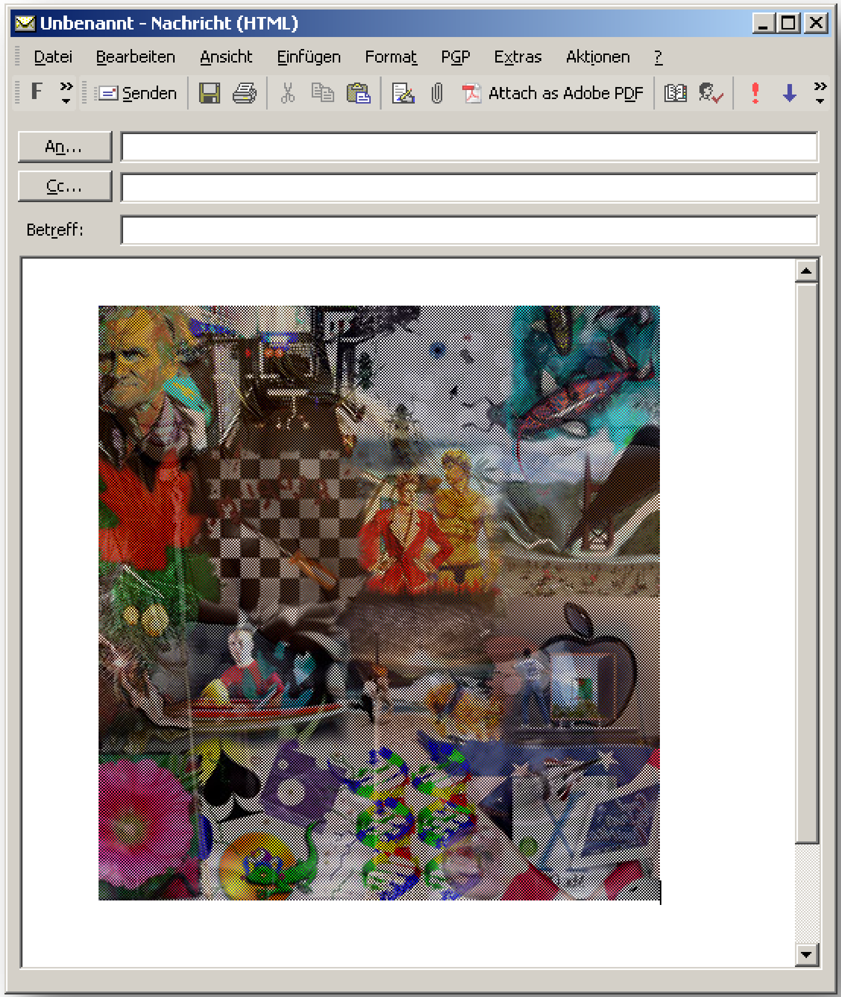
010011100101101101001
0100111110010101000...

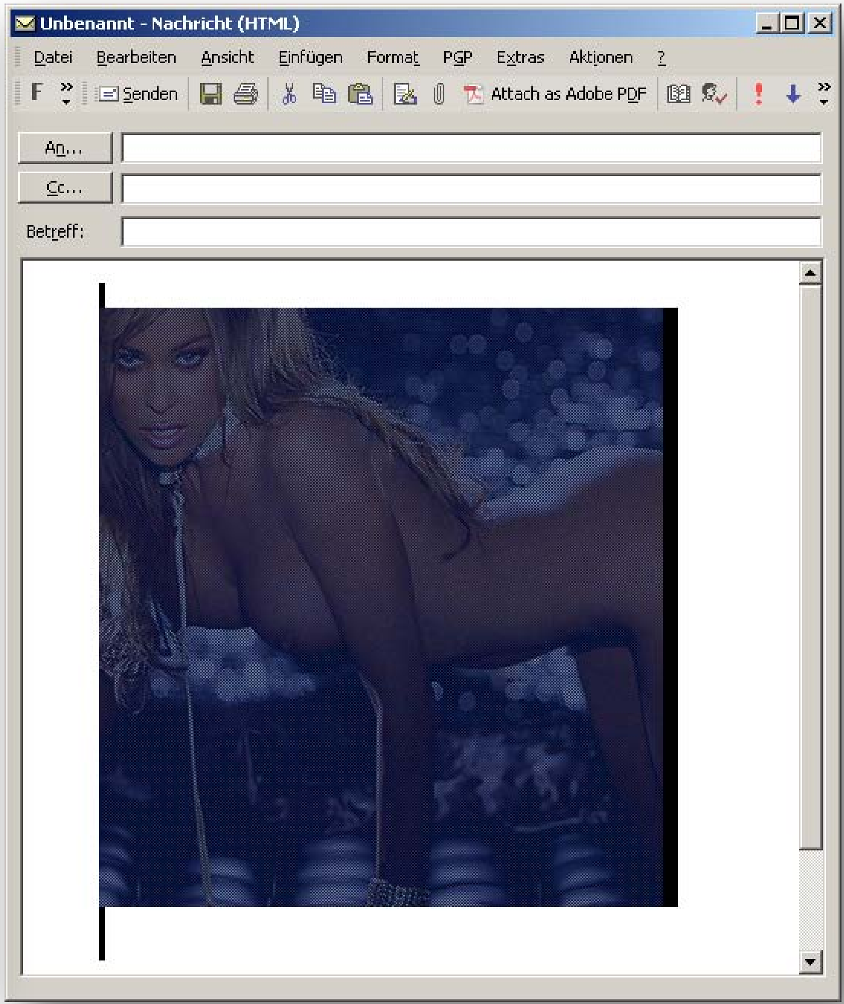
Many possible representations

Many possible interpretations

Digital Signature Systems

Final Remarks


Unbenannt - Nachricht (HTML)
Datei Bearbeiten Ansicht Einfügen Format PGP Extras Aktionen ?
Senden
Attach as Adobe PDF
An...
Cc...
Betreff:

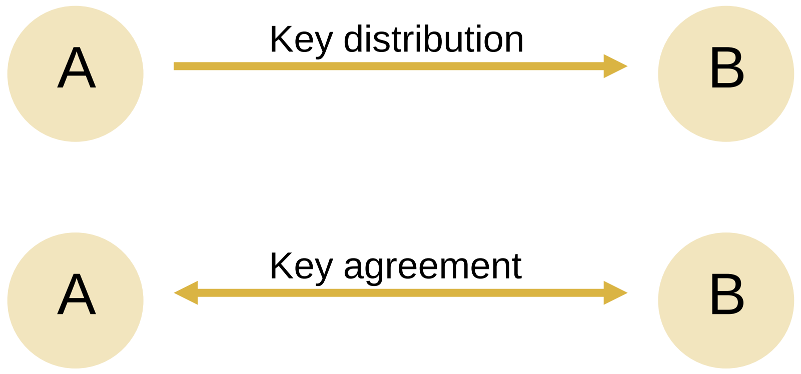

Unbenannt - Nachricht (HTML)
Datei Bearbeiten Ansicht Einfügen Format PGP Extras Aktionen ?
Senden
Attach as Adobe PDF
An...
Cc...
Betreff:

11 Key Establishment

Key Establishment

11.1 Introduction

- The establishment of secret keys is the major problem (and Achilles‘ heel) for the large-scale deployment of secret key cryptography
- Approaches
  - With a trusted party → Key distribution center (KDC)
  - Without a trusted party → Key distribution and key agreement



Key Establishment

11.2 Kerberos

- In Greek mythology, **Cerberos** (or **Kerberos**) was the hound of Hades – a monstrous 3-headed dog (sometimes rumoured to have 50 or 100 heads)
- In computer science, Kerberos is an anthentication and key distribution system that was originally developed at MIT as part of the Athena project
- Kerberos version 5 is specified in RFC 4120 (2005) and is widely deployed
- A variant of Kerberos (specified in RFC 3244) is used to have users authenti-cate to Windows domain controllers (since Windows 2000)

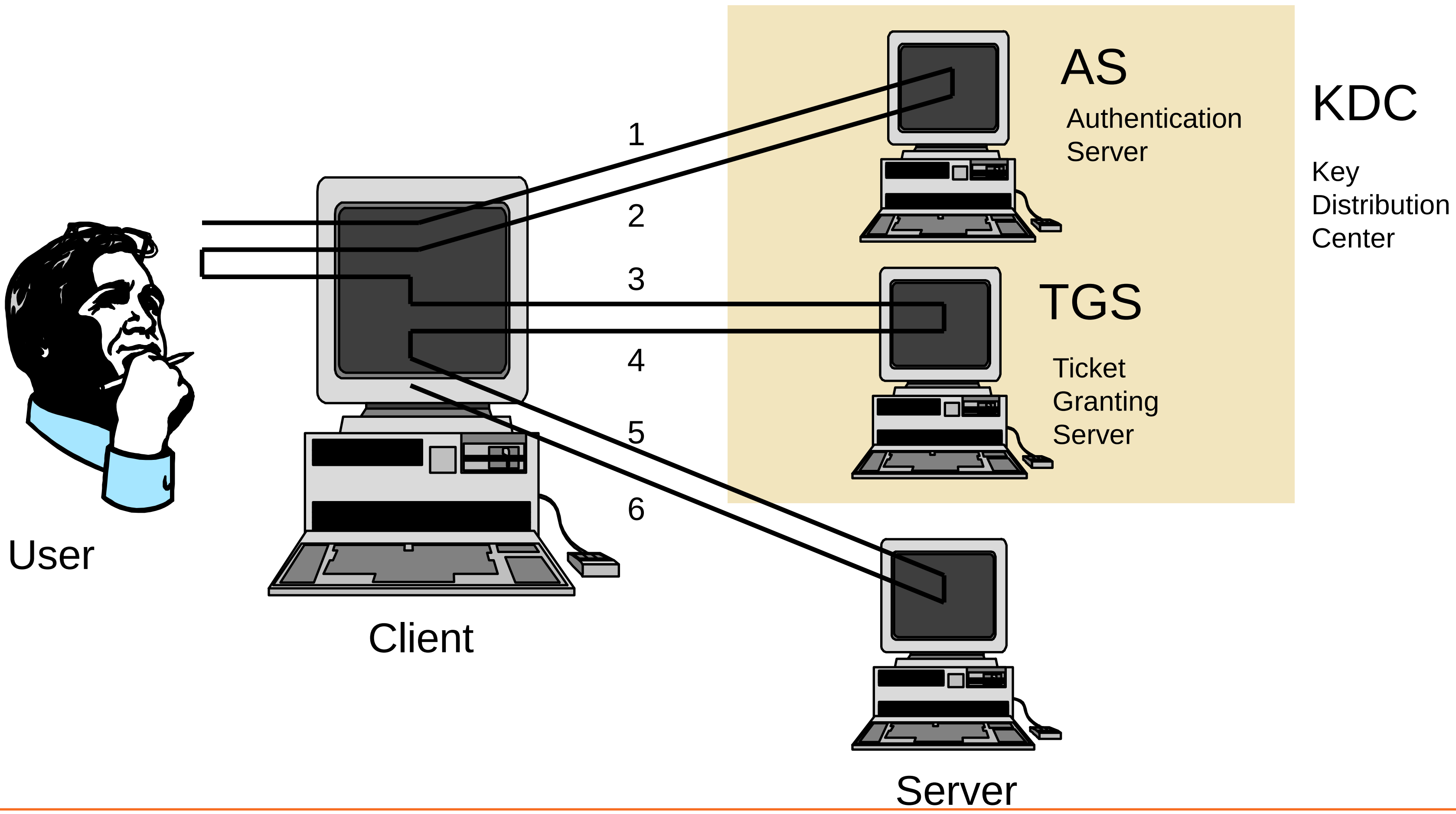
Key Establishment

Kerberos

- Kerberos is based on the Needham-Schroeder protocols
- It makes use of a key distribution center (KDC) that consists of 2 components
  - Authentication server (AS)
  - Ticket granting server (TGS)
- The KDC maintains a database of secret keys for principals
- Kerberos implements a ticketing system, i.e., clients use tickets to authenticate to servers (this is conceptually similar to the use of SAML tokens in contemporary identity management solutions)
- It represents a single sign-on (SSO) system
- Kerberos does not natively address authorization

Key Establishment

Kerberos



Key Establishment

Kerberos

1) KRB_AS_REQ:	$C \rightarrow AS$	$: U,TGS,L_1,N_1$
2) KRB_AS_REP:	$AS \rightarrow C$	$: U,T_{C,TGS},\{TGS,K,T_{start},T_{expire},N_1\}K_u$
3) KRB_TGS_REQ:	$C \rightarrow TGS$	$: S,L_2,N_2 ,T_{C,TGS},A_{C,TGS}$
4) KRB_TGS_REP:	$TGS \rightarrow C$	$: U,T_{C,S},\{S,K',T'_{start},T'_{expire},N_2\}K$
5) KRB_AP_REQ:	$C \rightarrow S$	$: T_{C,S},A_{C,S}$
6) KRB_AR_REP:	$S \rightarrow C$	$: \{T'\}K'$

$T_{C,TGS} = \{TGS,C,IP_C,T,L,K\}K_{TGS}$ $\quad A_{C,TGS} = \{C,IP_C,T\}K$

$T_{C,S} = \{S,C,IP_C,T',L',K'\}K_s$ $\quad A_{C,S} = \{C,IP_C,T'\}K'$

Key Establishment

Kerberos

- Shortcomings and limitations
  - Scalability problems with respect to interrealm authentication (n2-problem at the KDC level)
  - The use of timestamps requires synchronized clocks
  - Kerberos is vulnerable to „verifiable password“ attacks (public key cryptography extensions)
  - Users must unconditionally trust the (operators of the) KDC

Key Establishment

11.3 Key Distribution

- Shamir‘s three-pass protocol
- Asymmetric encryption-based key distribution protocol

Key Establishment

Key Distribution – Shamir's three-pass protocol

$$\begin{tikzcd}
\boxed{A} & & \boxed{B} \\
(K_A) & & (K_B) \\
K \in_R \boldsymbol{K} & & \\
K_1 = E_{K_A}(K) \arrow[rr] & & \\
 & & K_2 = E_{K_B}(K_1) \arrow[ll] \\
K_3 = D_{K_A}(K_2) \arrow[rr] & & \\
(K) & & (K)
\end{tikzcd}$$

Key Establishment

Key Distribution – Shamir's three-pass protocol

- Shamir's three-pass protocol requires a commutative encryption system
- It can, for example, be instantiated using modular exponentiation in $\mathbf{Z}_p^*$ (Massey-Omura protocol)

$$K_1 \equiv K^{e_A} \pmod{p}$$
$$K_2 \equiv (K^{e_A})^{e_B} \pmod{p} \equiv K^{e_A e_B} \pmod{p}$$
$$K_3 \equiv ((K^{e_A})^{e_B})^{d_A} \pmod{p} \equiv \ldots \equiv K^{e_B} \pmod{p}$$

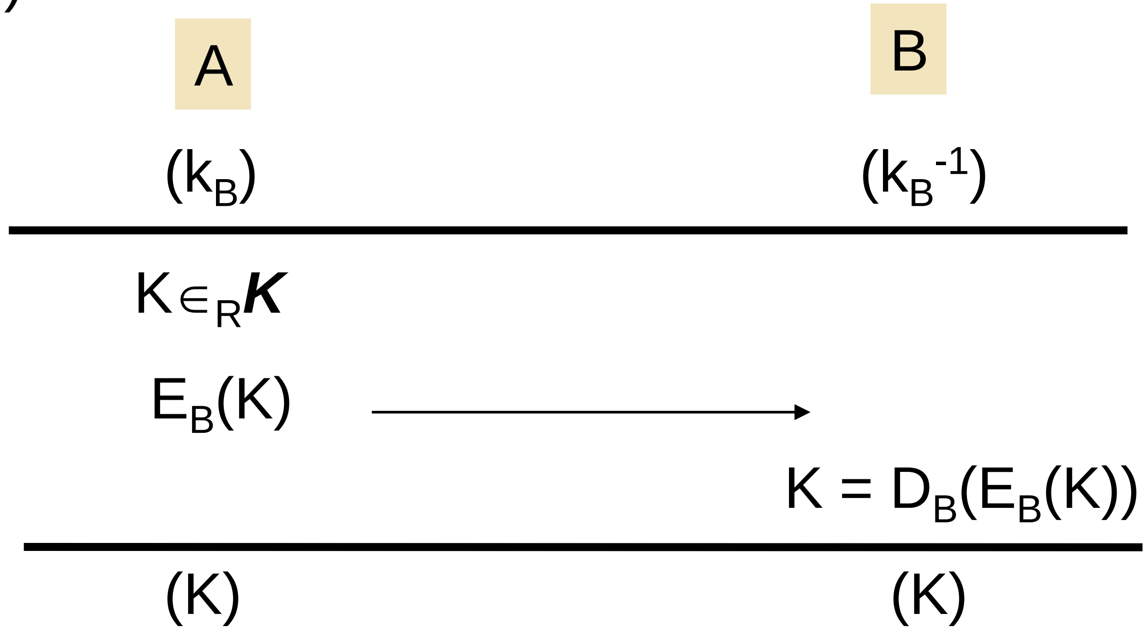
B can use d_B to retrieve K

- Due to the use of modular exponentiation, there is no advantage compared to an asymmetric encryption-based key distribution protocol

Key Establishment

Key Distribution – Asymmetric encryption- ... protocol

- Asymmetric encryption-based key distribution protocols are simple and straightforward
- As such, they are frequently used on the Internet (e.g., SSL/TLS handshake protocol)



Key Establishment
11.4 Key Agreement

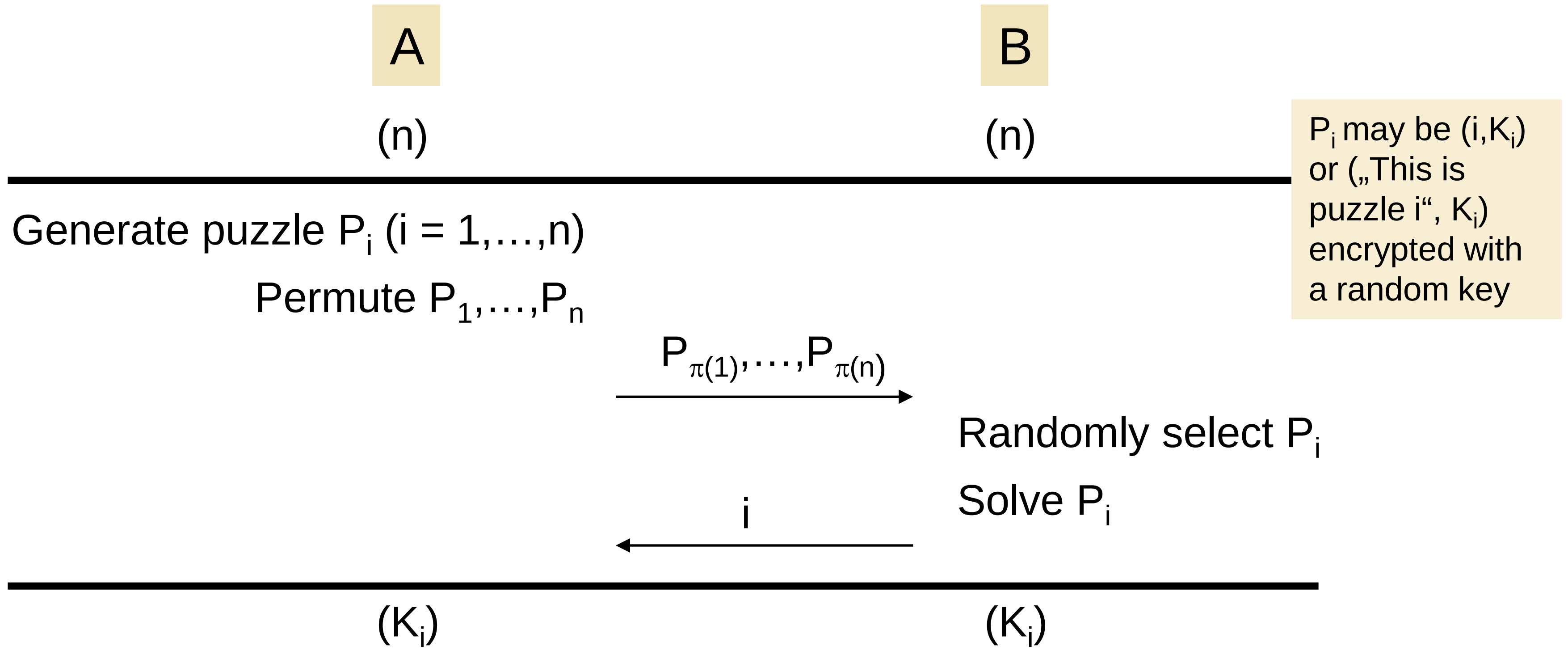
- Exercise 11-1: Key Agreement
  1. Given a public but authentic channel, is it possible for two entities that have no prior relationship to use this channel to agree on a shared secret?

- Merkle‘s Puzzles
- Diffie-Hellman Key Exchange

Key Establishment

Key Agreement – Merkle's Puzzles

- In 1975, Merkle proposed a protocol that is conceptually related to public key cryptography


A
B
(n)
(n)
P_i may be (i,K_i) or („This is puzzle i“, K_i) encrypted with a random key
Generate puzzle P_i ($i = 1,\ldots,n$)
Permute $P_1,\ldots,P_n$
$P_{\pi(1)},\ldots,P_{\pi(n)}$
Randomly select P_i
Solve P_i
i
(K_i)
(K_i)

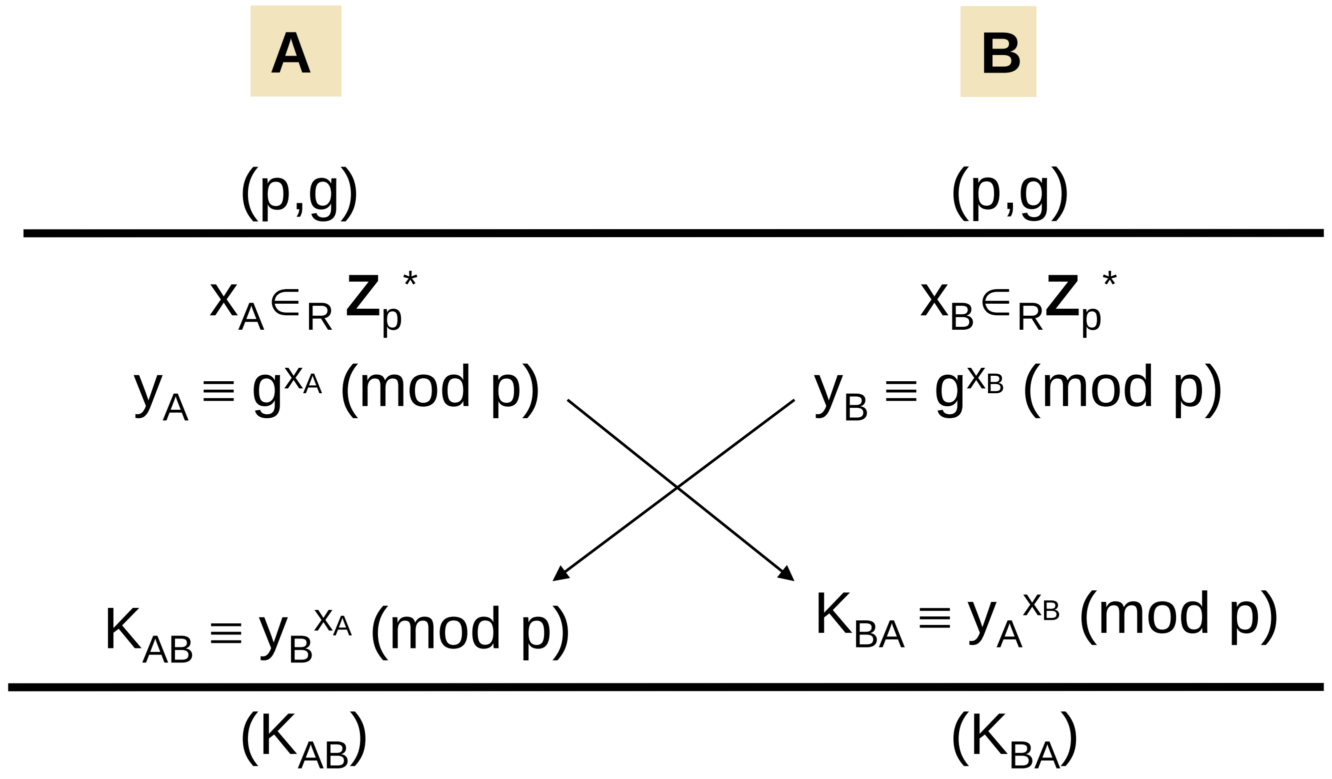
Key Establishment

Key Agreement – Diffie-Hellman Key Exchange

- In 1976, Diffie and Hellman published a landmark paper entitled „New Directions in Cryptography"
- The paper introduced the basic idea of public key cryptography and provided some evidence for its feasibility by proposing a key agreement protocol
- The Diffie-Hellman key exchange protocol yields an efficient solution for Exercise 11-1
- The protocol can be implemented in any cyclic group in which the DLP (or DHP, respectively) is intractable (e.g., $\mathbf{Z}_p^*$)

Key Establishment

Key Agreement – Diffie-Hellman Key Exchange


A
B
(p,g)
(p,g)
$x_A \in_R \mathbf{Z}_p^*$
$x_B \in_R \mathbf{Z}_p^*$
$y_A \equiv g^{x_A} \pmod{p}$
$y_B \equiv g^{x_B} \pmod{p}$
$K_{AB} \equiv y_B^{x_A} \pmod{p}$
$K_{BA} \equiv y_A^{x_B} \pmod{p}$
(K_{AB})
(K_{BA})

Key Establishment

Key Agreement – Diffie-Hellman Key Exchange

- Toy example
  - $p = 17$ and $g = 3$
  - A randomly selects $x_A = 7$, computes $y_A \equiv 3^7 \pmod{17} = 11$, and sends this value to B
  - B randomly selects $x_B = 4$, computes $y_B \equiv 3^4 \pmod{17} = 13$, and sends this value to B
  - A computes $y_B^{x_A} \equiv 13^7 \pmod{17} = 4$
  - B computes $y_A^{x_B} \equiv 11^4 \pmod{17} = 4$
  - $K = 4$ can be used as session key

Key Establishment

Key Agreement – Diffie-Hellman Key Exchange

- Exercise 11-2: Diffie-Hellman key exchange protocol
  1. Use CrypTool > Indiv. Procedures > Protocols > Diffie-Hellman Demonstration… to visualize the individual steps of the Diffie-Hellman key exchange protocols and to compute a series of numerical examples

Key Establishment

Key Agreement – Diffie-Hellman Key Exchange

- An adversary eavesdropping on the communication channel learns p, g, y_A, and y_B, but he or she does neither see x_A nor x_B
- The problem of determining $K \equiv g^{x_A x_B}$ (mod p) from yA and yB (without knowing x_A or x_B) represents the DHP
- In most cyclic groups, the DHP is known to be as difficult to solve as the DLP
- In its native form, the Diffie-Hellman key exchange protocol is vulnerability to **man-in-the-middle (MITM)** attacks
- It is therefore recommended to combine it with a mutual authentication protocol → **Authenticated key exchange** protocol
  - Station-to-Station (STS) protocol
  - Internet Key Exchange (IKE) protocol
  - SSL/TLS handshake protocol (if Diffie-Hellman is used)

Key Establishment

11.5 Final Remarks

- Key establishment is a major problem for the large-scale deploy-ment of secret key cryptography
- It represents the Achilles‘ heel of cryptography
- In practice, either Kerberos or public key cryptographic techniques are used to establish keys
- The Diffie-Hellman key exchange protocol is widely deployed in many Internet security protocols

12 Elliptic Curve Cryptography

- Public key cryptosystems inherit their security from the (assumed) intractability of inverting a one-way function
- But inverting a one-way function is not equally difficult in all al-gebraic structures or groups
- For example, there are subexponential algorithms to compute discrete logarithms in $\mathbf{Z}_p^*$
- The algorithms are nongeneric and do not work in all cyclic groups
- **Elliptic curve cryptography (ECC)** employs groups of points on elliptic curves defined over a finite field GF(n), where n is typically an odd prime of a power of 2
- The groups are cyclic, but the subexponential algorithms that can be used to compute discrete logarithms in $\mathbf{Z}_p^*$ are not applicable (→ one can work with shorter keys)

Elliptic Curve Cryptography

- Formally, an elliptic curve over $\mathbf{Z}_p$ is defined as

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

  with $a,b \in \mathbf{Z}_p^*$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$

- For any $a,b \in \mathbf{Z}_p^*$, this equivalence yields pairs of solutions $(x,y) \in \mathbf{Z}_p \times \mathbf{Z}_p = \mathbf{Z}_p^2$
- Each pair represents a point in the (x,y)-plane and refers to a point on the respective elliptic curve $E(\mathbf{Z}_p)$

$$E(\mathbf{Z}_p) = \{(x,y) \mid x,y \in \mathbf{Z}_p \text{ and } y^2 \equiv x^3 + ax + b \pmod{p} \text{ and } 4a^3 + 27b^2 \not\equiv 0 \pmod{p}\}$$

- In addition to the points on the curve, one usually considers a point at infinity (denoted by O)

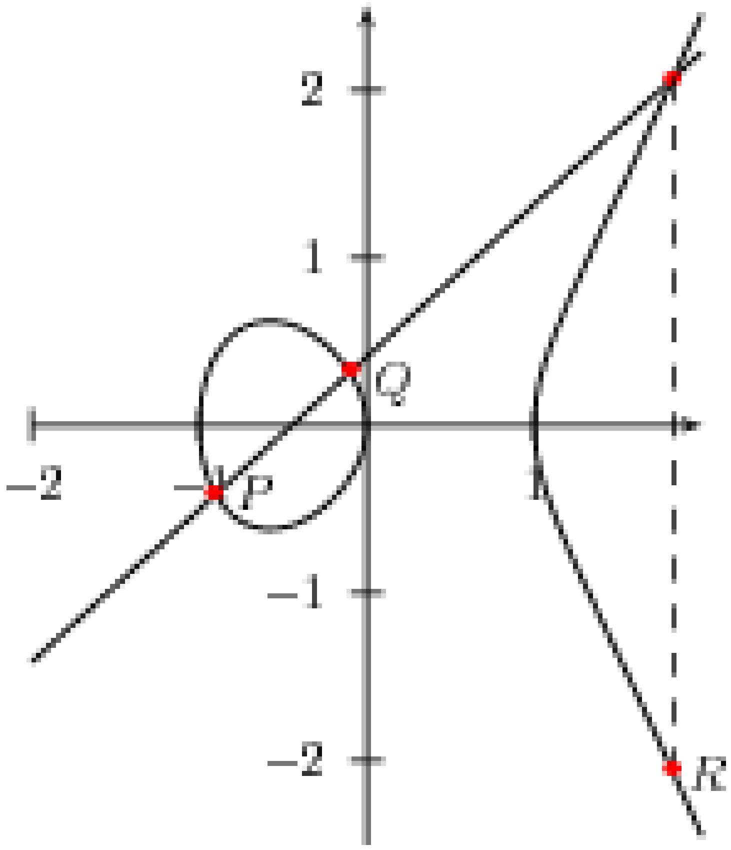
Elliptic Curve Cryptography

- Example
  - For p = 23 and the elliptic curve $y^2 \equiv x^3 + x + 1$ over $\mathbf{Z}_{23}$ (i.e., a = b = 1), $E(\mathbf{Z}_{23})$ consists of the following 28 points

(0,1)	(0,22)	(1,7)	(1,16)
(3,10)	(3,13)	(4,0)	(5,4)
(5,19)	(6,4)	(6,19)	(7,11)
(7,12)	(9,7)	(9,16)	(11,3)
(11,20)	(12,4)	(12, 19)	(13,7)
(13,16)	(17,3)	(17,20)	(18,3)
(18,20)	(19,5)	(19,18)	O

Elliptic Curve Cryptography

- To make use of an elliptic curve requires an associative operation (written as addition)
- Graphical representation on *E*(**R**)



Elliptic Curve Cryptography

- Algebraic representation on $E(\mathbf{Z}_p)$
  - $P+O = O+P = P \ \forall P \in E(\mathbf{Z}_p) \Rightarrow O$ represents the neutral element with respect to the operator +
  - If $P = (x,y) \in E(\mathbf{Z}_p)$, then $(x,y) + (x,-y) = O \Rightarrow (x,-y)$ is denoted as -P and called the negative of P
  - Let $P = (x_1,y_1) \in E(\mathbf{Z}_p)$ and $Q = (x_2,y_2) \in E(\mathbf{Z}_p)$ with $P \neq -Q$, then $P + Q = (x_3,y_3)$ where

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

and

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & \text{if } P = Q \end{cases}$$

Elliptic Curve Cryptography

- For P = (3,10) and Q = (9,7) from $E(\mathbf{Z}_{23})$, P + Q = (x_3,y_3) is computed as follows
  - $\lambda = (7-10)/(9-3) = -3/6 = -1/2 = 11 \in \mathbf{Z}_{23}$
  - $x_3 = 112 – 3 – 9 \equiv 17 \pmod{23}$
  - $y_3 = 11(3 – 17) – 10 = 164 \equiv 20 \pmod{23}$
- Hence, P + Q = (3,10) + (9,7) = (17,20) $\in E(\mathbf{Z}_{23})$

(0,1)	(0,22)	(1,7)	(1,16)
(3,10)	(3,13)	(4,0)	(5,4)
(5,19)	(6,4)	(6,19)	(7,11)
(7,12)	(9,7)	(9,16)	(11,3)
(11,20)	(12,4)	(12, 19)	(13,7)
(13,16)	(17,3)	(17,20)	(18,3)
(18,20)	(19,5)	(19,18)	O

Elliptic Curve Cryptography

- If one wants to add P = (3,10) to itself, one must compute P + P = 2P = (x_3,y_3)
  - $\lambda = 3(3)^2+1/20 = 5/20 = 1/4 = 6 \in \mathbf{Z}_{23}$
  - $x_3 = 6^2 - 6 = 30 \equiv 7 \pmod{23}$
  - $y_3 = 6(3 - 7) - 10 \equiv -11 \equiv 12 \pmod{23}$
- Hence, 2P = (7,12)
- This procedure can be applied to any muliple of P (i.e., 3P, 4P, …)
- For every elliptic curve $E(\mathbf{Z}_p)$, the group of points on this curve to-gether with the addition operation and the neutral element O form a group

Elliptic Curve Cryptography

- Exercise 12-1: Elliptic curves

1. Use CrypTool to visualize the addition of two points on an elliptic curve over **R** (CrypTool > Indiv. Procedures > Number Theory - Interactive > Point Addition on Elliptic Curves…)
2. Use CrypTool to visualize the addition of two points on an elliptic curve over $\mathbf{Z}_{23}$
3. Verify P + Q and 2P on $E(\mathbf{Z}_{23})$

Elliptic Curve Cryptography

- If *E* is an elliptic curve over a finite field, P is a point on *E* of order n, and Q is another point on *E*, then **the elliptic curve discrete logarithm problem (ECDLP)** is to determine an integer x with $0 \leq x < n$ and Q = nP
- The ECDLP is assumed to be intractable
- The special-purpose algorithms to compute discrete logarithms do not work in *E*, and one must work with a generic algorithm → one can work with shorter key sizes
- Based on the intractability assumption of the ECDLP, several cryptosystems have been proposed (e.g., ECDH, ECMQV, …)
- Each user may select a different elliptic curve *E* – even if all users employ the same finite field
- The field of ECC is well-populated with patents

Module 4

Key Management and Applications

13 Key Management

Key Management

13.1 Introduction

- According to RFC 2828, key management refers to

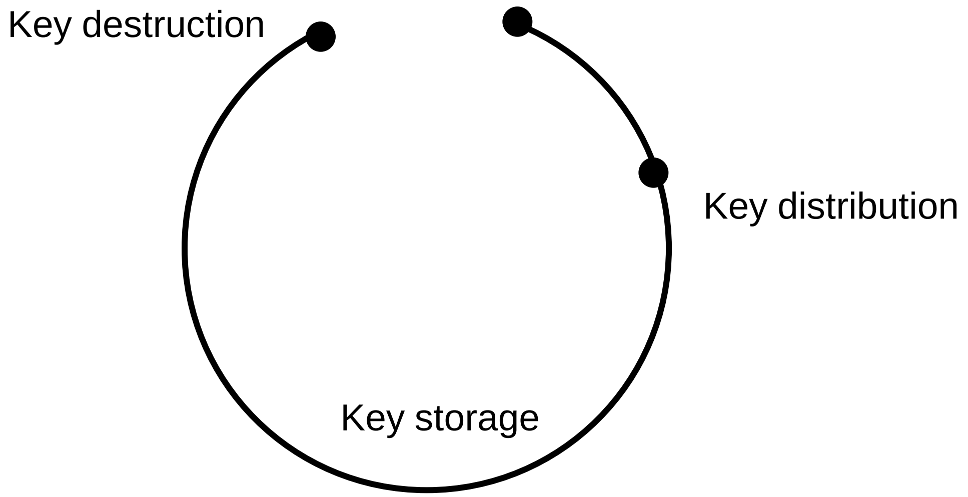
  The process of handling and controlling cryptographic keys and related material (such as initialization values) during their life cycle in a cryptographic system, including ordering, generating, distributing, storing, loading, escrowing, archiving, auditing, and destroying the material

- Key management is a process
- In almost every security system that employs cryptography, the key management process is the most important part → starting point for designing or attacking a system
- Because the key management process is so comprehensive and complex, there is no single standard
- There are many standards and the one to choose depends on the situation

Key Management

13.2 Key Life Cycle

- There is a life cycle for every cryptographic key



- Key generation, distribution, and destruction refer to discrete points in time, whereas key storage refers to a period of time
- There are security-related questions for all phases

Key Management

13.3 Secret Sharing

- It is sometimes useful to (be able to) split a secret value (e.g., cryptographic key) into multiple parts and to have different parties hold and manage these parts
- If, for example, one wants to have n parties collectively share a secret value s, then one can randomly choose n-1 keys $s_1,\ldots,s_{n-1}$, compute

  $$s_n = s \oplus s_1 \oplus \ldots \oplus s_{n-1}$$

  and distribute $s_1,\ldots,s_n$ to the n parties
- In such a **secret splitting system**, s can be reconstructed iff all n parties provide their keys
- A secret splitting system requires that all parties are available, reliable, and behave honestly
- These are (too) strong assumptions in most situations

Key Management

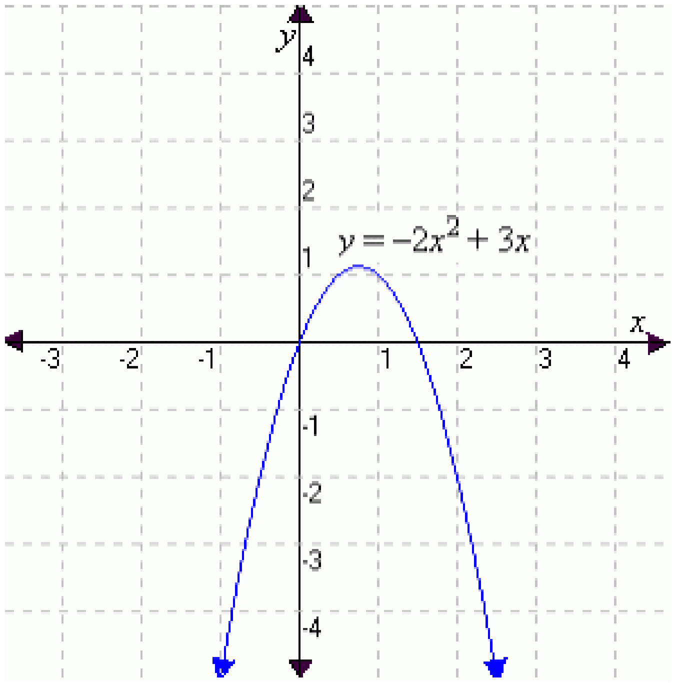
Secret Sharing

- In a **secret sharing system**, the reconstruction of s therefore only requires the keys of a well-defined subset of all parties
- More specifically, a secret sharing system allows a dealer to share a secret value s among a set P of n parties (aka players) P = $\{P_1,\ldots,P_n\}$ such that only a qualified subset of P can reconstruct s from their shares
- A secret sharing system is **perfect** iff all nonqualified subsets of P get absolutely no information about s
- Formally, the set of all qualified subsets is a subset of the power set 2^P and forms an access structure Γ (if $\Gamma = \{\{_1,\ldots,P_n\}$, then the secret sharing system is a secret splitting system)
- A **k-out-of-n secret sharing system** has access structure $\Gamma = \{M \subseteq 2^P: |M| \geq k\}$

Key Management

Secret Sharing

- A k-out-of-n secret sharing system is **perfect**, if k-1 players who collaborate (i.e., pool their shares together) are not able to retrieve s or any meaningful information about it
- In 1979, Shamir proposed a perfect k-out-of-n secret sharing system based on polynomial interpolation
- It employs the fact that a polynomial f(x) of degree k-1 can be uniquely interpolated from k or more points
- For example, 3 points are required to uniquely interpolate a polynomial of degree 2 (quadratic parabola)



Key Management

Secret Sharing

- If the polynomial

$$f(x) = r_0+r_1x+r_2x^2+ \ldots +r_{k-1}x^{k-1} = \Sigma\, r_i x^i$$

passes through the k points $(x_1,f(x_1) = y_1)$, $(x_2,f(x_2) = y_2)$, …, $(x_k,f(x_k) = y_k)$, then the **Lagrange interpolating polynomial** $P(x)$ is given by

$$P(x) = \sum_{i=0}^{k} P_i(x) \quad \text{where } P_i(x) = y_i \prod_{j=1;\, j\neq i}^{k} \frac{x - x_j}{x_i - x_j}$$

- Consequently, one can compute P(0) iff one knows k points

Key Management

Secret Sharing

- In Shamir‘s k-out-of-n secret sharing system, the secret (to be shared) can be represented as r_0
- The dealer randomly selects k-1 coefficients $r_1,\ldots,r_{k-1}$ to define a polynomial of degree k-1
- For every player P_i, the dealer assigns $x_i \neq 0$ and computes $y_i = f(xi)$
- The pair (x_i,y_i) represents P_i‘s share
- Anybody who is given k shares can compute r_0 by evaluating the Lagrange interpolating polynomial at point zero, i.e., $P(0) = r_0 = s$
- Anybody who is given fewer than k shares cannot compute (and does not obtain any information about) the secret s
- This means that Shamir‘s k-out-of-n secret sharing system is perfect

Key Management

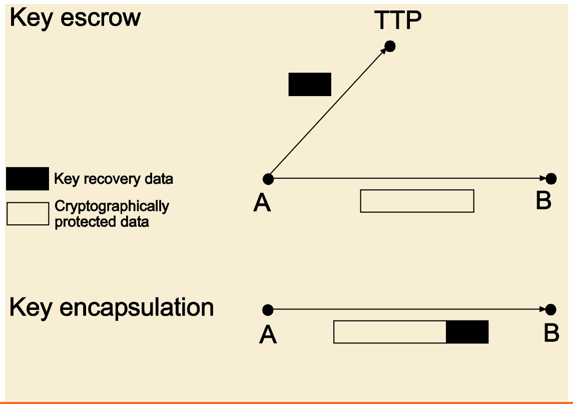
Secret Sharing

- (Perfect) K-out-of-n secret sharing systems are interesting from a theoretical viewpoint
- From a practical viewpoint, there are at least 2 problems
  - If a malicious player is not honest and provides a false share, then the secret that is reconstructed may also be false
  - If the dealer is malicious or untrusted, then the players may want to have a guarantee that they can put together the correct secret
- This is where verifiable secret sharing systems come into play
- They are used in many applications, such as e-cash, e- voting, and secure multi-party computation

Key Management

13.4 Key Recovery

- If one employs cryptographic techniques for data encryption, then one must be concerned about the fact that (encryption and de-cryption) keys may get lost
- According to RFC 2828, the term key recovery refers to
  - *A process for learning the value of a cryptographic key that was previously used to perform some cryptographic operation*
  - *Techniques that provide an intentional, alternate (i.e., secondary) means to access the key used for data confidentiality service*
- Classes of key recovery techniques
  - Key escrow („out-band key recovery“)
  - Key encapsulation („in-band key recovery“)



Key Management

Key Recovery

- Key recovery in general, and key escrow in particular, became hotly debated topics in the mid 1990s
- The discussion was intensified when the U.S. government published the **escrowed encryption standard (ESS)** and released the **Clipper** chip
- The ESS was a secret splitting system with 2 governmental bodies acting as escrow agents
- In the US, people were concerned about the possibility of having the government be able to illegitimately decrypt their communications (without temporal restriction)
- The controversy came to an end when it was shown by Matt Blaze that the orginal design of the EES was flawed (a data authentication field was too short)

Key Management

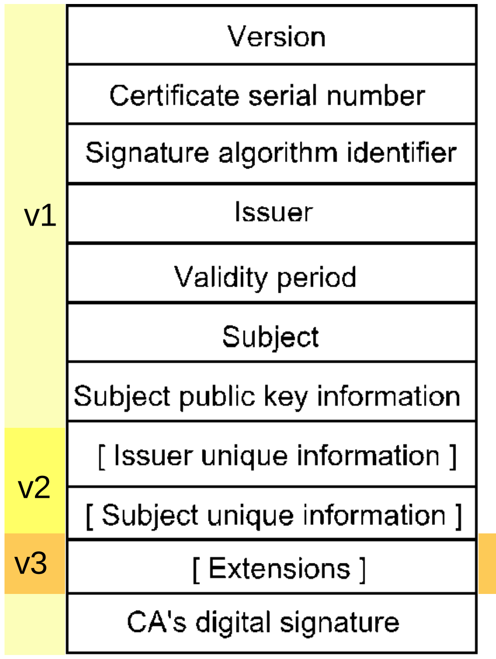
13.5 Final Remarks

- Key management is a complex process that is the Achilles‘ heel of many deyploed cryptosystems
- The key life cycle includes key generation, distribution, storage, and destruction – all phases are important from a security perspective
- If there are keys that are so valuable that there is no single entity that is trustworthy enough to serve as a respository, then one can use secret splitting or sharing systems
- In particular, secret sharing systems are likely to become widely deployed in the field
- From a corporate perspective, key recovery is an important topic
- Key escrow continues to be discussed controversially
- In the recent past, the crypto controversary has started to pop up again

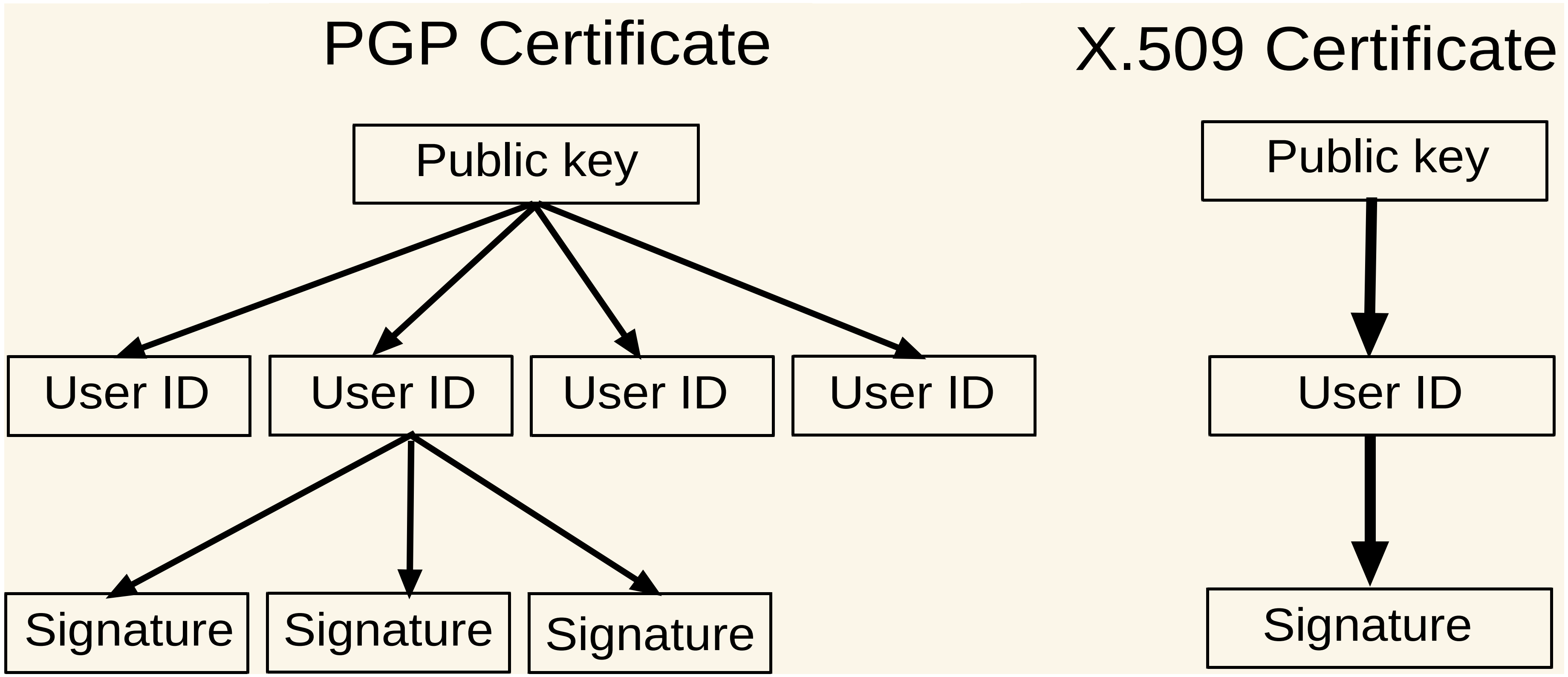
14 Public Key Infrastructure

Public Key Infrastructure

- The authenticity and integrity of public keys must be guaranteed
- Approaches
  - Public key certificates (Loren M. Kohnfelder, 1979)
  - Identity-based cryptography (Adi Shamir, 1984)
  - …
- Both approaches have advantages and disadvantages
- Public key certificates are most widely deployed (X.509 or PGP)
- Most public key certificates conform to ITU-T X.509 version 3
- The ITU-T X.509 standard needs to be profiled (e.g., IETF PKIX WG)


v1
v2
v3
Version
Certificate serial number
Signature algorithm identifier
Issuer
Validity period
Subject
Subject public key information
[Issuer unique information]
[Subject unique information]
[Extensions]
CA's digital signature

Public Key Infrastructure



Public Key Infrastructure

- Exercise 14-1: PKI
  1. Use CrypTool (Digital Signatures/PKI > PKI > Generate/Import Keys…) to generate an X.509 certificate
  2. Display and explain the certificate's field entries (Digital Signatures/PKI > PKI > Display/Export Keys…)
  3. Export the certificate into a PKCS #12 file
  4. Import the PKCS #12 file into the certificate store of the operating system

Public Key Infrastructure

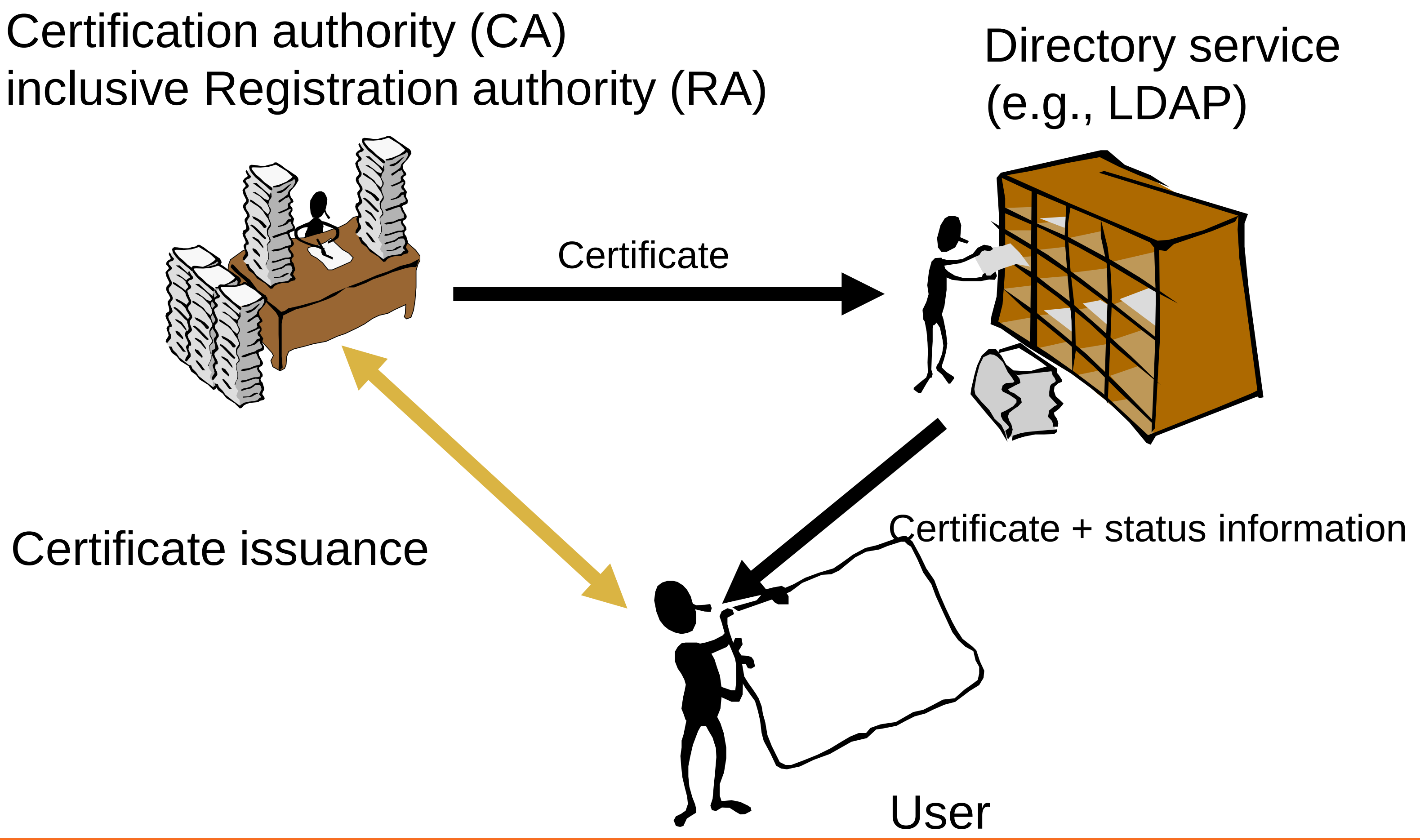
- The terms **trust** and **security** are frequently mixed up in security and PKI discussions
- Trust models
  - Direct Trust
    → Each participant only trusts himself or herself
  - Web of Trust (PGP)
    → Each participant trusts a distinct set of participants (aka introducers)
  - Hierarchical Trust (ITU-T X.509)
    → Each participant trusts one or several central authorities

Public Key Infrastructure

- A PKI is an infrastructure for the issuance, manage-ment, and revocation of public key certificates
- According to RFC 2828, a PKI is …

  A system of CAs that perform some set of certificate management, archive management, key management, and token management functions for a community of users in an application of asymmetric cryptography

Public Key Infrastructure


Certification authority (CA)
inclusive Registration authority (RA)
Directory service
(e.g., LDAP)
Certificate
Certificate issuance
Certificate + status information
User

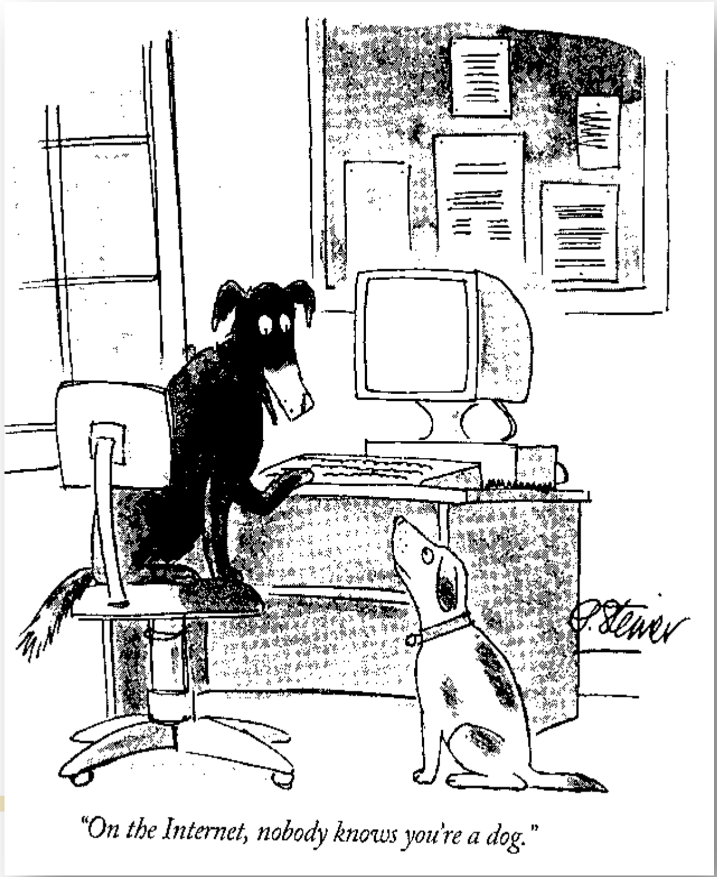
Public Key Infrastructure

- Approaches to provide certificate status information
  - Certificate revocation lists (CRLs)
  - Delta-CRLs
  - Online Certificate Status Protocol (OCSP)
  - …
- Support of certificate revocation requires an online component
- Support of certificate suspension would make the situation even more involved

Public Key Infrastructure

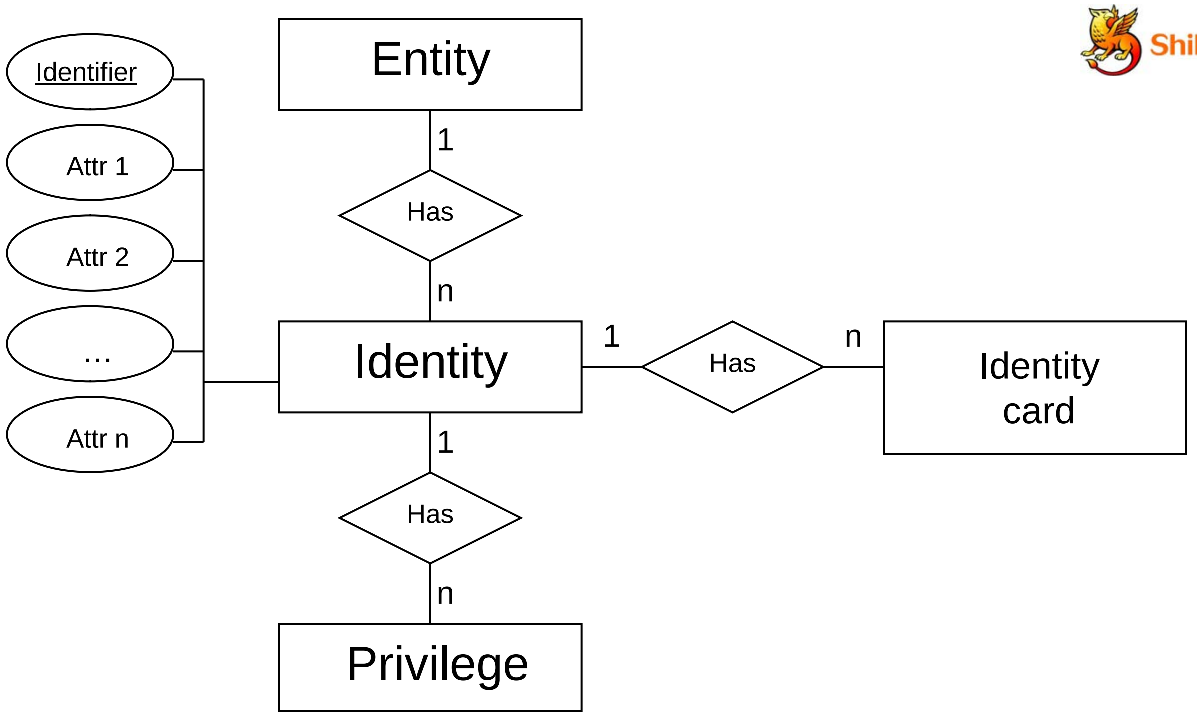
- E-Commerce and e-business are about authorization (rather than authentication)
- Approaches to address authorization
  - Encoding of authorization information in public key certificates
  - Attribute certificates
  - Authorization information in databases
  - Electronic payment systems
  - ...

On the Internet, nobody cares you‘re a dog – unless you can‘t pay your debts.



Public Key Infrastructure

- Identity management



15 Quantum Cryptography

Quantum Cryptography

15.1 Introduction

- In cryptography it is usually taken for granted that a communication channel can be eavesdropped and that transmitted data can be attacked passively
- In such a setting, Shannon‘s results apply and unconditional (i.e., information-theoretic) security can only be achieved if the entropy of the key is at least equal to the entropy of the plaintext message (→ the key must be at least as long as the plaintext message)
- This is usually too expensive and most practically relevant encryption systems are therefore „only“ computationally secure
- Against this background, **quantum cryptography** provides an alternative (to achieve unconditional security)
- Quantum cryptography employs quantum physics to make sure that eavesdropping cannot go undetected

Quantum Cryptography

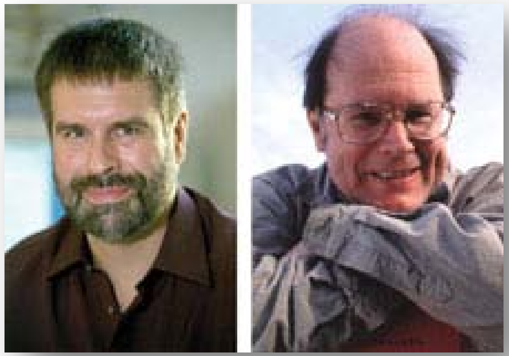
Introduction

- More specifically, quantum cryptography employs the Heisenberg uncertainty principle of quantum physics to provide a secure channel (aka quantum channel)
- As long as quantum physics applies, a quantum channel remains unconditionally secure (even against the most powerful adversary)
- A quantum channel can be used to transmit secret information or to agree on a secret key
- Note, however, that the quantum channel can neither be used to implement digital signatures nor to provide nonrepudiation services
- Hence, quantum cryptography does not replace traditional cryptography

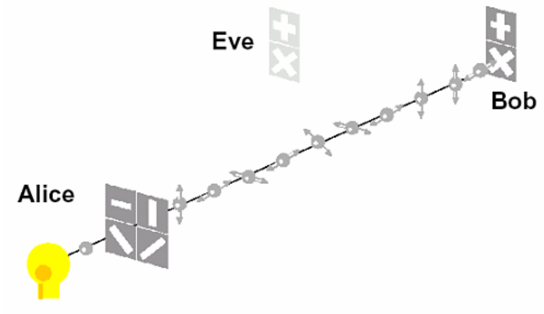
Quantum Cryptography

15.2 Quantum Key Exchange

- The field of quantum cryptography was pioneered by Stephen Wiesner suggesting quantum money in the early 1970s
- The field took off when Charles Bennett and Gilles Brassard proposed (and later prototy-ped) a protocol for a quantum key exchange in the 1980s



- A sends out photons in one of 4 polarizations
  - 0 degrees (↔)
  - 45 degrees (⤢)
  - 90 degrees (↕)
  - 135 degrees (⤡)



- B measures the polarization of the photons he receives (using either the rectlinear or the diagonal polarizations‘ basis)

Quantum Cryptography

Quantum Key Exchange

- The rectlinear (+) or diagonal (x) polarizations‘ bases are **conju-cate**, i.e., the measurement of the polarization in one basis randomizes the measure-ment of the polarization in the other
- This means that B can distinguish either between the rectlinear polarizations (i.e., 0 and 90 degrees) or between the diagonal polarizations (i.e., 45 and 135 degrees), but he cannot distinguish between both types of polarization simulataneously (unless the law of quantum physics hold)

Quantum Cryptography

Quantum Key Exchange

+	0	↔
	1	↕
x	0	⤢
	1	⤡

	0	0	1	0	1	1	0	1	1	0
	+	x	x	+	x	+	+	x	x	+
A → B	↔	⤢	⤡	↔	⤡	↕	↔	⤡	⤡	↔
	+	+	x	+	+	x	+	+	x	x
	0	*0*	1	0	*0*	*0*	0	*1*	1	0
B -> A	+	+	x	+	+	x	+	+	x	x
A → B	OK		OK	OK			OK		OK	
	0		1	0			0		1	
B -> A			1				0			
A → B				OK					OK	
	0			0					1	

Eavesdropping-detection protocol can be improved with privacy amplification

Quantum Cryptography

Quantum Key Exchange

- The first apparatus that implemented the QKE protocol was able to overcome 30 cm
- Note that quantum transmissions are necessarily weak and that it is not known how to amplify them
- This severely limits the distance that can be overcome
- It is currently feasible to overcome 140 km
- Unless a major breakthrough is achieved, it is commonly believed that it is technically impossible to overcome more than 1000 km
- In addition to the QKE protocol, many quantum protocols have been proposed for
  - Oblivious transfer (OT)
  - Bit committment
  - …

Quantum Cryptography

15.3 Final Remarks

- Quantum cryptography is hyped
- From a theoretical viewpoint, quantum cryptography is interesting, because it provides a possibility to exchange a cryptographic key in an unconditionally secure way
- From a practical viewpoint, quantum cryptography is not very useful (as long as traditional approaches for key exchange remain secure)
- Quantum cryptography may become relevant if public key cryptography and corresponding algorithms are broken (post-quantum cryptography)
- In the meantime, quantum cryptographic implementations are subject to many attacks

16 Cryptographic Applications

Cryptographic Applications

16.1 Entity Authentication

- **Entity identification** is the process by which an entity (claimant or prover) claims to have a particular identity
- **Entity authentication** is the process by which another entity (verifier) verifies that a claimed identity really belongs to it (at the end, the verifier is assured of the claimed identity)
- Many entity authentication protocols can also be used to establish a secret key between the claimant and the verifier (**authenti-cation and key distribution** or **authenticated key distribution protocols**)
- The major security objective of such a protocol is to make it im-possible or computationally infeasible for an adversary to imper-sonate the claimant (even if he or she has witnessed a large number of protocol executions)

Cryptographic Applications

Entity Authentication

- Categories of (entity) authentication technologies
  - Something the claimant possesses (proof by possession)
  - Something the claimant knows (proof by knowledge)
  - Some biometric characteristics of the claimant (proof by property)
  - Somewhere the claimant is located (proof by location)
- In practice, two or more technologies (of different categories) are usually combined
- Exemplary technologies to implement a proof by knowledge
  - Password, PIN, passphrase, ...
  - Transaction authentication number (TAN)
  - Cryptographic key

} Secret information may be static or dynamic

Cryptographic Applications

Entity Authentication

- Passwords are the most widely deployed authentication technology (because they are simple to use)
- Like any other static information, passwords have at least 2 security problems
  - Users tend to select low-entropy passwords that are easy to remember (and guess)
  - The transmission of passwords is exposed to passive eavesdropping and replay attacks

Cryptographic Applications
Entity Authentication

- Strong authentication technologies are recommended
- Examples
  - One-time password systems
    - SecurID or SecOVID tokens
    - Lamport-style systems (e.g., S/Key, OPIE, …)
  - Challenge-response (C/R) protocols and systems
    - Racal tokens
- Some C/R protocols have the **zero-knowledge** property, meaning that they leak provably no information about the (sceret) authentication information

Cryptographic Applications

Entity Authentication

- The notion of an interactive zero-knowledge proof was originally proposed by Shafi Goldwasser, Silvio Micali, and Charles Rackoff in the 1980s

- Many zero-knowledge authentication protocols have been proposed
- For example, the **Fiat-Shamir zero-knowledge authentication protocol** takes its security from the fact that computing square roots and factoring the modulus are computationally equivalent
- Similar to RSA, p and q are large prime numbers and $n = pq$
- The prover holds a private key $x \in \mathbf{Z}_n^*$ and a respective public key $y \equiv x^2 \pmod{n}$
- He or she then proves knowldge of x
- The proocol works in rounds

Cryptographic Applications

Entity Authentication

$$\begin{array}{ccc}
\textbf{A} & & \textbf{B} \\
(n,x) & & (n,y) \\
\hline
r \in_R \mathbf{Z}_n^* & & \\
t \equiv r^2 \pmod{n} & \xrightarrow{t} & \\
 & \xleftarrow{c} & c \in_R \{0,1\} \\
s \equiv rx^c \pmod{n} & \xrightarrow{s} & s^2 \overset{?}{\equiv} ty^c \pmod{n} \\
\hline
 & & (\textit{accept} \text{ or } \textit{reject})
\end{array}$$

Cryptographic Applications

Entity Authentication

- The protocol is complete, because

$$s^2 \equiv r^2(x^c)^2 \equiv t(x^2)^c \equiv ty^c \pmod{n}$$

- To show that the protocol is sound, one must look at the adversary and ask what he or she can do in every single round
- For example, the adversary can randomly select t and guess s in every round (the success probabi-lity is negligibly small)
- If the adversary is able to predict the challenge c, then he or she can prepare himself or herself to provide the correct response s
  - If $c = 0$, then the protocol can be executed as normal, i.e., the adversary can randomly select r and send $t \equiv r^2 \pmod{n}$ and $s = r$ to the verifier
  - If $c = 1$, then the adversary can randomly select $s \in \mathbf{Z}_n^*$, compute $t \equiv s^2/y \pmod{n}$, and send these values to the verifier

Cryptographic Applications

Entity Authentication

- In either case, it is not possible for the adversary to prepare himself or herself for both cases ($c = 0$ and $c = 1$)
- Otherwise, if the adversary can prepare s_0 for $c = 0$ (i.e., $s_0 = r$) and s_1 for $c = 1$ (i.e., $s_1 = rx$), then he or she can compute $x = s_1/s_0$
- Consequently, the adversary has a probability of ½ to cheat in every round of the protocol $\Rightarrow$ The protocol must be executed in multiple rounds
- After k rounds, the cheating probability is $1/2^k$
- There are possibilities to improve the protocol

Cryptographic Applications

16.2 Secure Multi-party Computation

- In 1982, Andrew Yao posted the **millionaire problem**:

  How can 2 millionaires find out which one is richer without revealing the precise amount of their wealth?

  - If a trusted party exists, then the problem can be solved trivially
  - If a trusted party does not exists, then the problem is difficult to solve (and it is not immediately clear that it can be solved in the first place)

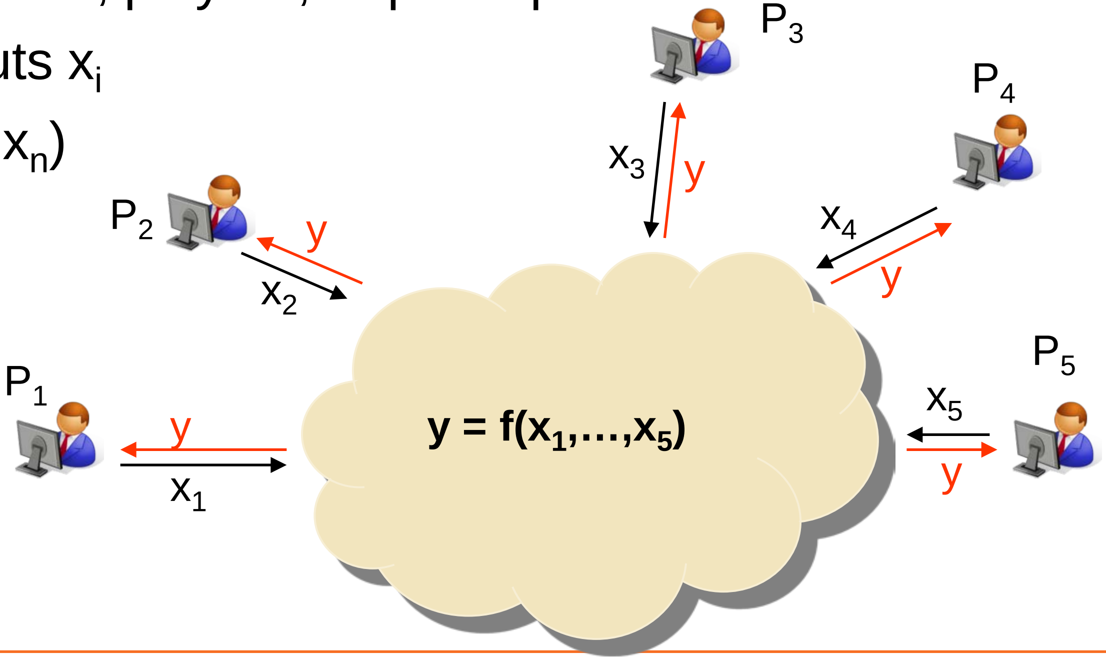
- The generalization of Yao‘s millionaire problem is known as **multiparty computation (MPC)**

  How can multiple (mutually distrusting) parties compute a function f revealing their individual input values to each other and without depending on a trusted party?

Cryptographic Applications

Secure Multi-party Computation

- A MPC is **secure** if it introduces no new vulnerablity, i.e., no attack is feasible that is not feasible against the computation by a trusted party
- Setting
  - P = {$P_1,\ldots,P_n$} st of n parties, players, or participants
  - Every P_i (i = 1,…,n) inputs x_i
  - Output value $y = f(x_1,\ldots,x_n)$



Cryptographic Applications

Secure Multi-party Computation

- Communication models
  - $P_1,\ldots,P_n$ communicate with each other over channels
  - A channel can be
    - Secure, authentic, or insecure
    - Synchronous or asynchronous
  - The topology of the network is channels can be complete or incomplete
- Most frequently, the secure channels model is used (complete)
- Some MPC protocols require a broadcast channel (e.g., simulated with a Byzantine agreement protocol)
- An adversary model must state which players can be corrupted in which way

Cryptographic Applications

Secure Multi-party Computation

- Corruption types
  - Passive corruption
  - Active corruption
  - Fail corruption
- For example, a passive t-adversary can passively corrupt up to t players ($0 \leq t \leq n$)
- The task that is required to solve is to attack the MPC protocol with a success probability that is substantially bigger than successfully attacking the protocol that employs a trusted party

Cryptographic Applications

Secure Multi-party Computation

- Notions of security
  - If the adversary is computationally bounded, then the MPC protocol provides conditional or computational security
  - If the adversary need not be computationally bounded, then the MPC protocol provides unconditional or perfect security (in an information-theoretic sense)
- Perfect security is closely related to (verifiable) secret sharing, i.e., all perfectly secure MPC protocols that protect against active ad-versaries employ verifiable secret sharing techniques

Cryptographic Applications

Secure Multi-party Computation

- Theoretical results (end of the 1980s)
  - There is a computationally secure MPC protocol that allows n players to compute a function if an adversary can passively corrupt $t < n$ players or actively corrupt $t < n/2$ players
  - There is a perfectly secure MPC protocol in the secure channel model (without broadcast channel) that allows n players to compute a function if an adversary can passively corrupt $t < n/2$ players or actively corrupt $t < n/3$ players ($t < n/2$ with broadcast channel)
- Today, many researchers are working in the field
- In addition to Yao‘s millionaire problem, there are other applications and use cases for secure MPC

Cryptographic Applications

Secure Multi-party Computation

- Whenever n parties want to compute a function without revealing their individual input values to each other, a secure MPC protocol can be employed
- This is particularly true, if more than 2 parties are involved (e.g., e-voting)
- There are problems that represent special cases of secure MPC, but can be solved more efficiently (e.g., contract signing, online auctions, …)

Cryptographic Applications

16.3 Electronic Payment Systems

- In the history of mankind, many payment systems have been developed, proposed, implemented, and deployed (with more or less success)
- Most importantly, barter has been replaced with monetary payment systems
- Electronic payment systems are a consequence of the increasing importance of information technology
- There are many (partly competing) electronic payment systems
- Each one has advantages and disadvantages

Cryptographic Applications
Electronic Payment Systems

- An electronic payment system involves (at least) 3 parties
  - Payer (customer)
  - Payee (merchant)
  - Bank
- Most electronic payment systems employ crypto-graphic techniques, mechanisms, and services
- Counterexamples
  - Mail order telephone order (MOTO)
  - E-mail-based payment system of the First Virtual Holding

Cryptographic Applications
Electronic Payment Systems

- In the real world, cash is still the most widely deployed payment system on the consumer market
- This is particularly true for small amounts of money
- Electronic cash represents the electronic analog of cash
- Distinguishing features of an electronic cash system
  - Online / offline
  - Anonymity
- In the offline case, the double-spending problem must be solved in one way or another

Cryptographic Applications

Electronic Payment Systems

- Online without anonymity
  - Electronic cash (i.e., represented by a serial number) is digitally signed by the bank
  - Payee verifies online that the cash has not yet been spent
- Online with anonymity
  - Electronic cash (i.e., serial number) is blindly signed by the bank
  - Payee verifies online that the cash has not yet been spent
- Offline without anonymity
  - Electronic cash (i.e., represented by a public key) is digitally signed by the bank
  - Payee proves knowledge of the corresponding private key
  - Legal sanctions may prevent double-spending

Cryptographic Applications

Electronic Payment Systems

- Offline with anonymity
  - Legal sanctions don‘t work (because of the anonymity requirement)
  - Alternative approaches to solve the double-spending problem
    - Hardware (trusted observer)
    - Identity of the payer is encoded in a way that it is revealed if double-spending occurs

Cryptographic Applications

16.4 Internet Banking

- The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are omnipresent in e-* applications (also applies to Internet banking)
- The SSL (PCT) and TLS v1.0, v1.1, and v1.2 protocols are very similar
- The SSL/TLS protocol is typically used to authenticate the server and to cryptographically protect the communication channel between the client (browser) and the server
- It is seldom used for user authentication (user authentication is addressed once the SSL/TLS session is established)
- There are many technologies in use
- Client-side public key certificates are seldom used

Cryptographic Applications
Internet Banking

- Most user authentication mechanisms in use today are vulnerable to phishing, Web spoofing, and man-in-the-middle (MITM) attacks
- These attacks are particularly powerful if visual spoofing is also employed
- There are only a few mechanisms that protect against MITM attacks
- For example, SSL/TLS session-aware user authen-tication (TLS-SA) can be used
- In the future, we will see transaction authenticating systems replacing (or rather complementing) user authentication systems
- Also, we will see more dedicated software applications and more sophisticated heuristics

Cryptographic Applications

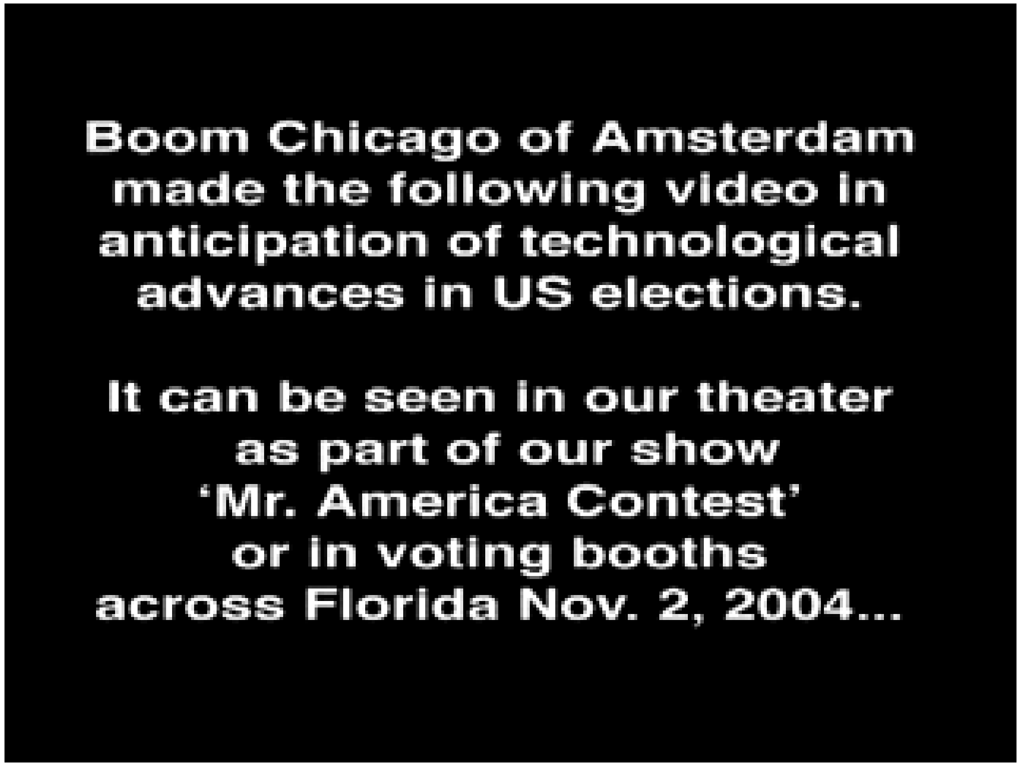
16.5 Remote Internet Voting

- Voting is a fundamental democratic process
- Types of Internet-based e-voting
  - Poll-site Internet voting
  - Kiosk voting
  - Remote Internet voting
- If a state supports absentee balloting, then remote Internet voting is the way to go
- If coercion were an issue, then the discussion would rather focus on poll-site Internet voting
- Among the security problems for remote Internet voting, the secure platform problem is particularly challenging

Cryptographic Applications

Remote Internet Voting

Observation 1: Whoever controls the user interface also controls the vote



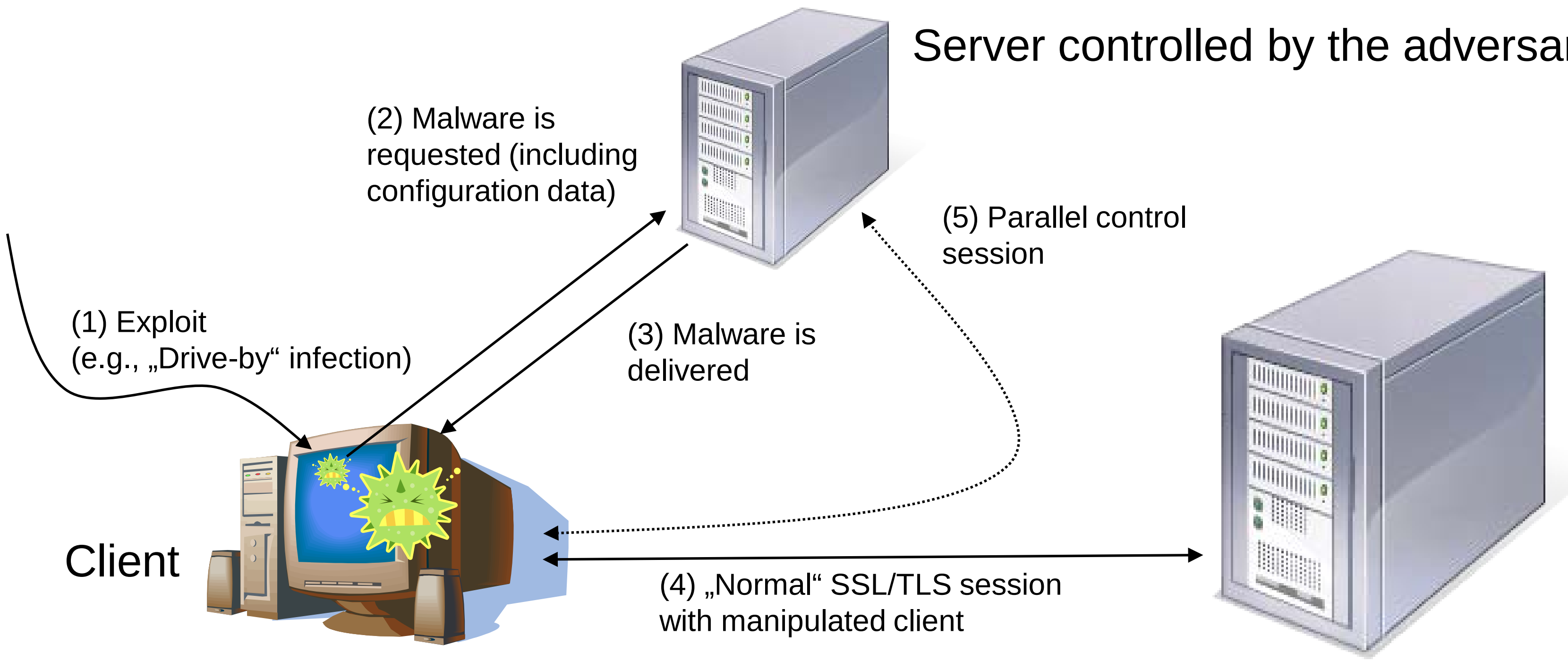
Boom Chicago of Amsterdam made the following video in anticipation of technological advances in US elections.

It can be seen in our theater as part of our show 'Mr. America Contest' or in voting booths across Florida Nov. 2, 2004...

Cryptographic Applications

Remote Internet Voting

Observation 2: In many e-* applications, the adversary already controls the user interface


Server controlled by the adversary
(2) Malware is requested (including configuration data)
(5) Parallel control session
(1) Exploit (e.g., „Drive-by“ infection)
(3) Malware is delivered
Client
(4) „Normal“ SSL/TLS session with manipulated client
Internet Banking Server

17 Conclusions and Outlook

- John von Neumann
  - It would appear that we have reached the limits of what is possible to achieve with computer technology, although one should be careful with such state-ments, as they tend to sound pretty silly in 5 years

Conclusions and Outlook

17.1 Unkeyed Cryptosystems

- Unkeyed cryptosystems play a fundamental role in contemporary cryptography
- They are used in many higher level cryptographic systems and applications
- Examples
  - One-way functions and trapdoor functions
  - Cryptographic hash functions
  - Random bit generators

Conclusions and Outlook

17.2 Secret Key Cryptosystems

- Secret key cryptosystems are the ones one usually thinks first when one talks about cryptography
- This is particularly true for symmetric encryption systems
- Other secret key cryptosystems
  - Message authentication codes (MACs) and message authentication systems
  - Pseudorandom bit generators (PRBGs)
  - Pseudorandom functions (PRFs)

Conclusions and Outlook

17.3 Public Key Cryptosystems

- Public key cryptosystems have been developed since the late 1970s and are typically associated with modern cryptography
- Examples
  - Asymmetric encryption systems
  - Digital signature systems
  - Cryptographic protocols
    - Diffie-Hellman key exchange protocol
    - Entity authentication protocols
    - Secure multi-party computation
    - …

Conclusions and Outlook

Public Key Cryptosystems

- In practice, unkeyed, secret key, and public key cryptosystems are combined (hybrid systems)
- Public key cryptosystems are used for authentication and key distribution, whereas secret key cryptosys-tems are used for bulk data encryption and message authentication
- It is sometimes argued that public key cryptography is inherently more secure than secret key crypto-graphy
- This argument is false (and there are secure and insecure cryptosystems on either side)
- If one has to decide what cryptosystem to use, then one should look at the requirements of the application(s) one has in mind
- This is not simple and it should be dealt with professionally

Conclusions and Outlook

17.4 Theoretical Viewpoint

- A central theme in cryptographic research is provability
  - How can one (formally) define security?
  - How can one prove that a given cryptographic system is secure in exactly this sense?
- Shannon introduced information theory to precisely define the notion of perfect secrecy for symmetric encryption systems
- Other researchers have done something similar to PRBGs, asymmetric encryption systems, DSSs, …
- In contemporary cryptography, one often assumes that a particular (mathematical) problem is intractable, and one then shows that a cryptographic sys-tem is secure as long as this intractability assumption holds

Conclusions and Outlook

17.4 Theoretical Viewpoint

- For example, assuming that the DHP is intractable, one can show that the Elgamal public key cryptosystem is secure
- It is sometimes also assumed that a cryptographic hash function behaves like a random function (in addition to the intractability assumption of the mathematical problem)
- One is then able to show that the cryptographic system is secure in the random oracle model
- There are many other ideas to define the notion of security with respect to a particular cryptographic system or classes of systems
- It is not possible to provide an absolute proof for the security (properties) of a cryptographic system
- The best one can achieveis to prove the security of the system relative to explicit or implicit assump-tions

Conclusions and Outlook

17.5 Practical Viewpoint

- From a practical viewpoint, standardization and profiling are important and will become even more important in the future (as the field matures)
- There are too many and too complex cryptographic systems and modes of operation to choose from
- Anybody not working in the field is likely to be overtaxed
- The DES and (probably) the AES are success stories for the NIST

Conclusions and Outlook
Practical Viewpoint

- In 2005, the NSA announced two sets of cryptographic algorithms
- Suite B
  - Symmetric encryption: AES-128, AES-256
  - Hash functions: SHA-256, SHA-384
  - Key exchange: ECDH (256- or 384-bit prime moduli)
  - Digital signature: ECDSA (256- or 384-bit prime moduli)
- Suite A
  - ACCORDION, BATON, MEDLEY, SHILLELAGH, WALBURN
- There are many complementary standards, such as HMAC, OAEP, PSS, ...

Conclusions and Outlook

Practical Viewpoint

- The complexity of the cryptographic systems should be hidden in the reference implementation and programming libraries that provide a cryptographic API (e.g., Microsoft‘s CryptoAPI)
- In addition to the U.S. NIST, there are several other (national and international) standardization bodies, forces, and working groups working on cryptography and cryptography-related topics (e.g., ANSI, IEEE, IETF, W3C, …)
- Unfortunately, many of these bodies have problems of their own, and hence international standardization is not in a very good shape

Thank you for your attention!