DIMACS Security & Cryptography Crash Course
Lecture 1: Principles & Encryption

Prof. Amir Herzberg

Computer Science Department, Bar Ilan University

http://amir.herzberg.name

Agenda

- What is security?
  - Arbitrary adversary principle
  - Kerckhoffs' `law` – don't assume secret design
- Encryption: ciphers and cryptosystems
  - From early ciphers to one time pad
  - Perfect (unconditional) secrecy
  - Stream ciphers and pseudo-random bit generators
  - (Pseudo) Random Permutations as block ciphers
  - Practical block ciphers and their security
  - Minimal Assumptions Principles and cryptanalysis tolerance
  - Modes of operation of block ciphers
  - Encryption schemes (cryptosystems)
  - Cryptosystem security under Indistinguishability test
  - CPA-IND secure cryptosystem from PRP (block cipher)
- Cryptographic constructions and proofs in general
- Conclusions and summary of principles

What is `security`?

- Consider multiple parties (entities, agents)
- With (often) adversarial interests
- Ensure (some) interests of some parties
  - Often viewed as preventing threats / risks
- How?
  - Discourage adversarial behavior
    - Education, Punishment, Incentives
  - Prevent damage in spite of adversarial behavior
- This is very general - economy, legal,…
- Let’s focus on information (computer) science…

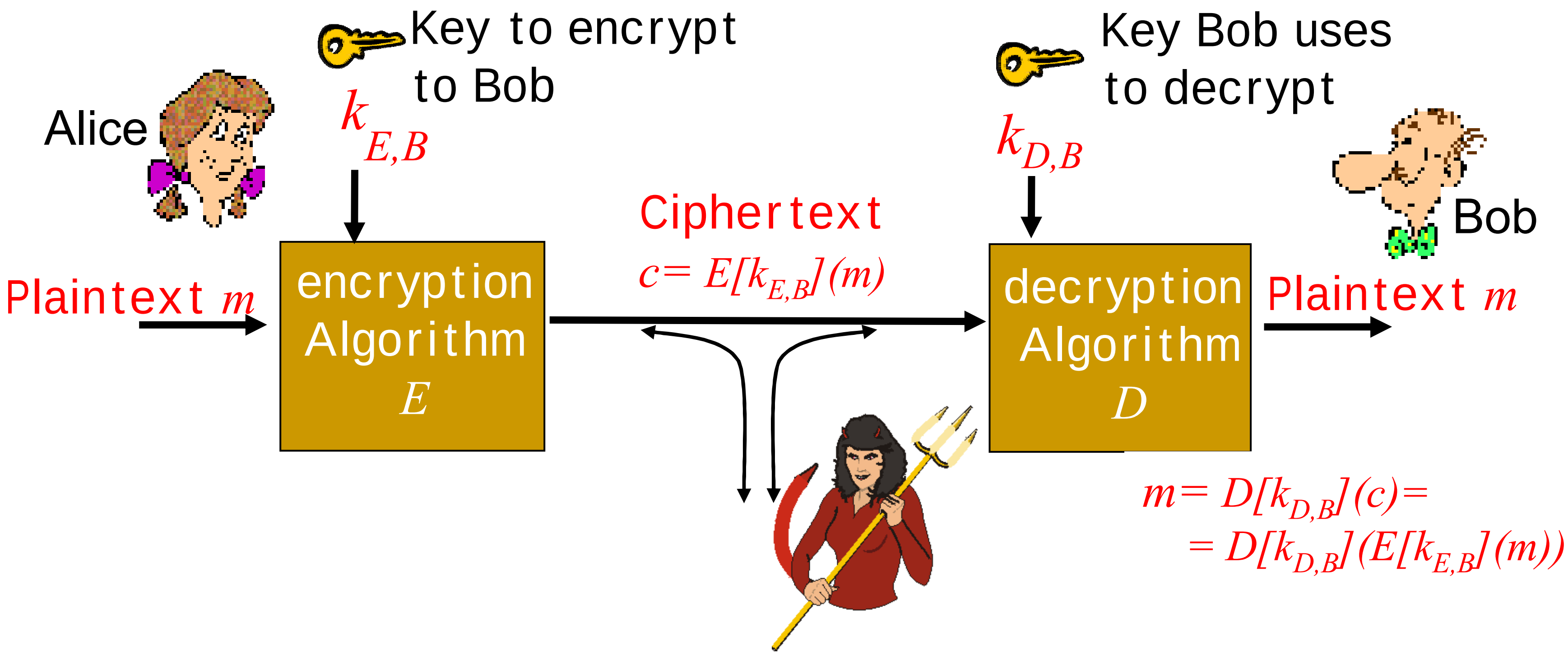
What is `security` for information?

- Discourage adversarial behavior
  - By providing proof (e.g. to court)
  - By appropriate incentives (mechanism design)
  - By reputation (reviews, history) – more later (PKI)
- Prevent damage in spite of adversarial behavior
  - ***Arbitrary Adversary Principle:*** Assume restrictions on capabilities of adversary – not on adversary's strategy!
  - Computational restrictions – limited computational abilities, e.g. speed, memory, …
  - Access restrictions (can't use console, can't change OS, can't read memory – in particular *keys*…)
  - Can we assume the adversary does not know the design??

Kerckhoff’s Principle: Known Design

- Attacking (e.g. cryptanalysis) of unknown design can be much harder
- But using non-secret designs…
  - No need to replace system once design is exposed
  - No need to worry that design was exposed
  - Establish standards for multiple applications:
    - Efficiency of production and of test attacks / cryptanalysis
- ***Kerckhoff’s Known Design Principle***: adversary knows the design – everything except the secret keys

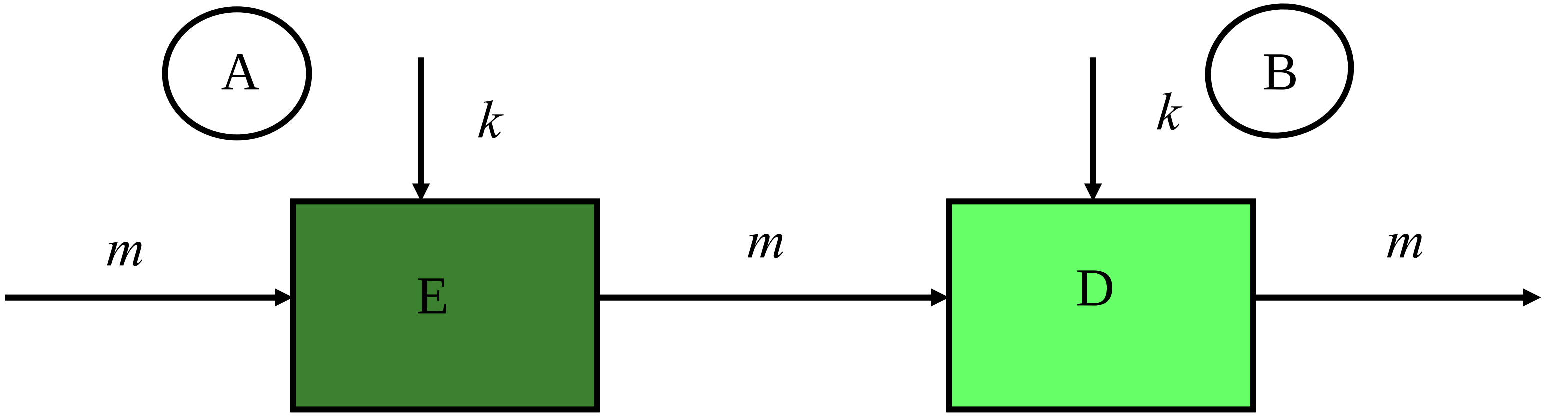
Consider Encryption…



symmetric key crypto: shared secret key ($k_{E,B}=k_{D,B}$) - today

public-key crypto: *public* encryption key $k_{E,B}$, matching *private* decryption key $k_{D,B}$ - tomorrow

Symmetric (shared secret key) Encryption



- Encryption: transforming secret message (*plaintext*) into garbled *ciphertext*
- Adversary should not learn anything about plaintext even if it gets ciphertext
- Classic goal of security / cryptography
  - In fact cryptography = secret writing
  - Predates computers… used by Romans and earlier
- Let's begin with some (simple) examples

Early Encryption

- Early encryption used mono-alphabetic ciphers
  - A set $\{<E_k,D_k>\}$ of permutation + inverses: $m=D_k(E_k(m))$
  - Such that the input (and output) domain is the set of characters
  - This is special case of block ciphers (here block=character)
- At-Bash cipher:
  - Jeremiah 51, 41: "איך נלכדה ששך . . ."
  - The word – ששך refers to – בבל (Babylon) by simple letter substitution: א ↔ ת, ב ↔ ש …
  - Namely: substitute first letter of alphabet (א) by last (ת), second letter (ב) by one-before-last (ש), etc…
- Used here probably for political reasons – afraid to say Babylon explicitly… No proof for `real` use for secrecy
- But we do know Ceasar used ciphers…

Caesar Cipher

- Rotate the 26 letters of the alphabet by 3:

- As formula:

$$c = E(p) = p+3 \pmod{26}$$

- The secrecy is in the algorithm (!!!!)
- There is one key (fixed permutation)
- Trivial to decipher (if algorithm is known)
  - But even if not known... Kerckhoff is right!

Keyed Caesar Cipher

- Rotate letters by key k, where $0 \le k < 26$.
- As formula: $c = E_k(p) = p + k \pmod{26}$
- Exhaustive Key Search Attack: try all (26) possible keys...
- Not very secure...
- ***Sufficient key length Principle:***
  - Number of possible keys should be large enough
  - To make attacks infeasible, using best adversary resources (HW) expected during `sensitivity period` of data
  - Using exhaustive key search or other feasible attacks
- Idea: use each key to encrypt just one char!

Monoalphabetic Substitution Cipher

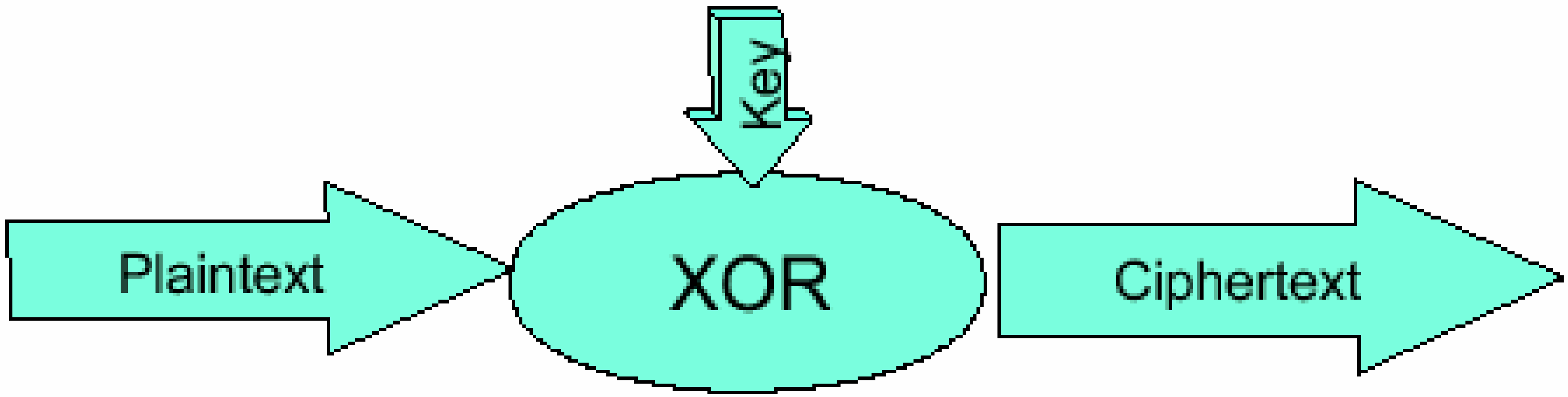
- Map each letter to some other letter (mapping is the key)
- $C_i = K[P_i]$
- Many keys (26! for 26 letters – more than 2^{80})
- Attack using letter distribution statistics (E leads - 13%)
- *Statistical attack*
  - *Identify letters by their frequencies, e.g. Pr('E')=13%*
  - *Also called Ciphertext only attack.*
- Need a better idea…

One-time Pad (Caesar) Cipher

- Idea 2: use each key to encrypt just one char!
- Use just the 26 permutations $c=p+k$…
- But use different key for each letter: $c_i=p_i+k_i$
- Therefore: $Pr(p_i=`A`|c_i=`B`)=Pr(k_i=1)=1/26$
  - Assuming $Pr(p_i=`A`)=1/26$ for a moment
- In fact for every p_i, c_i hold $Pr(p_i|c_i)=Pr(p_i)$
- →Plaintext gives no info about message
- Even if adversary has unbounded time
- As long as each key is chosen randomly
- More common: Bitwise One-Time Pad…

Bit-wise One Time Pad

- Each ciphertext bit is XOR of plaintext & key
  - $p_i \in \{0,1\}$, $k_i \in \{0,1\}$, $c_i = p_i \oplus k_i$
- Each key bit used only once
- Requires infinite secret shared random key
- Requires perfect synchronization to decrypt
- Shannon [S49]: unconditional secrecy…



Unconditional (Perfect) Secure Cipher

- Information-theory definition of secrecy (by Shannon)
- Let M be (finite) set of plaintext messages, $Pr(m)$ probability of message $m \in M$
- Let K be the key space, $Pr(k)$ probability of key $k \in K$
  - Usually uniform – $Pr(k)=1/|K|$
- A *cipher* is a set $\{E_k\}$ of permutations over M
- A cipher $\{E_k\}$ is <u>unconditionally (perfectly) secure</u> if for every $m' \in M$ and distribution on M holds: $Pr(m'=m|E_k(m))=Pr(m'=m)$, for random k and m.
- One-time Pad is perfect cipher… but requires $|K|=|M|$
  - Can we use same key k for two (or more) messages: $c_{2i}=p_{2i}+k_i$, $c_{2i+1}=p_{2i+1}+k_i$?
  - No, bad idea… known ciphertext attack: given $k_i = c_{2i} - p_{2i}$
  - Is there some short-key cipher (same key for multiple messages)?

Shannon's Perfect Security Theorem

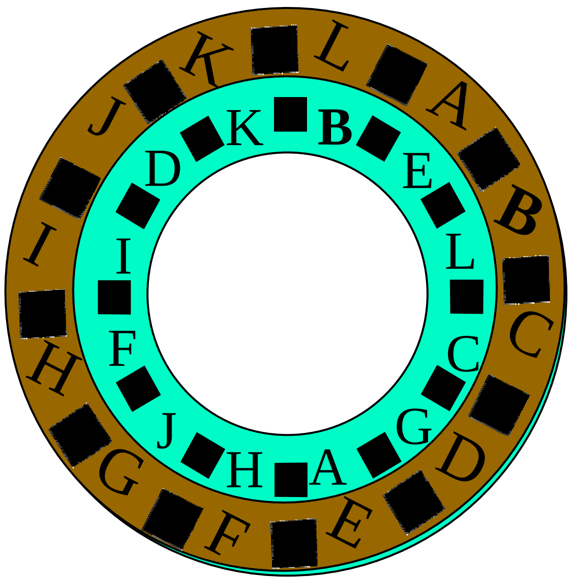
- Theorem [Shannon]: If cipher *E* is unconditionally secure, then $|K|\geq|M|$
- Hence: one time pad is as efficient as possible
- Sometimes, this is not a problem: setup long enough key in advance
- But often it is difficult to setup such long key:
  - Motivating Shannon to call this `theoretical secrecy`
  - Establish key between parties via the network
  - Support high-bandwidth and/or use secure – but limited – key storage
  - Collecting necessary randomness for the key

Collecting Randomness

- Surprisingly hard
- Physical devices/chips (e.g., sample radio): expensive, slow, unavailable to software
- Measuring human actions: slow and requires human interaction
- Both: bits often biased and dependent
- More common solution: use *stream cipher* - key changes every bit/byte/block

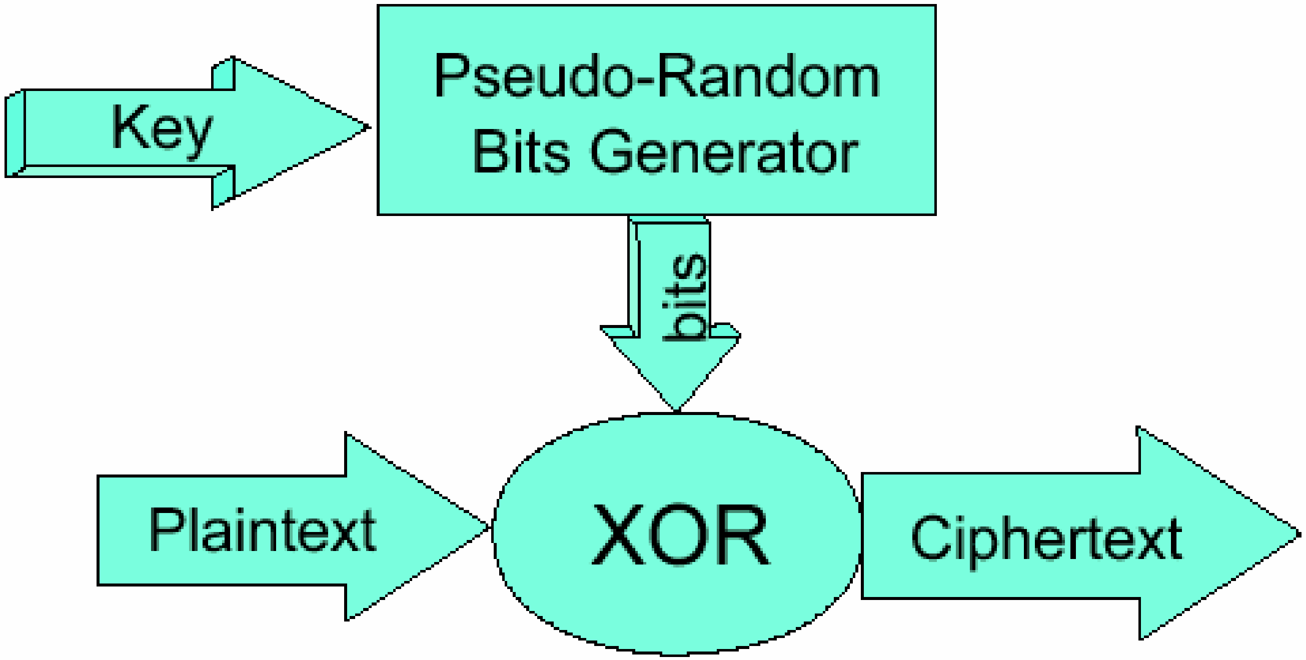
Early Stream Ciphers: Polyalphabetic

- Polyalphabetic ciphers: use character substitution but with changing keys (\`alphabets\`)
- Vigenére's cipher: Monoalphabetic substitution shifted, e.g. one position per letter: $C_i=K[i+P_i]$
  - Statistical attack harder, since different positions are used
  - Long message – can use statistics for repeating positions → **Limited key usage principle**
  - *Known Plaintext* attack: if attacker has encryption of any message, she can easily find out the substitution (known shift)
    - Plaintext: *BACK*
    - Ciphertext: *LLAL*
    - *K[B]=L, K[E]=A*
  - *Chosen Plaintext* attack: even easier if attacker can choose…
    - Plaintext: *AAAA…*
    - Ciphertext: *ELCG… = K[A], K[B], K[C], …*



Pseudo-Random Bits Generator (PRG) based stream cipher

- PRG is a function that given short random string (seed), creates long stream which is indistinguishable from random bits
- Computationally secure (beware of snake-oil)

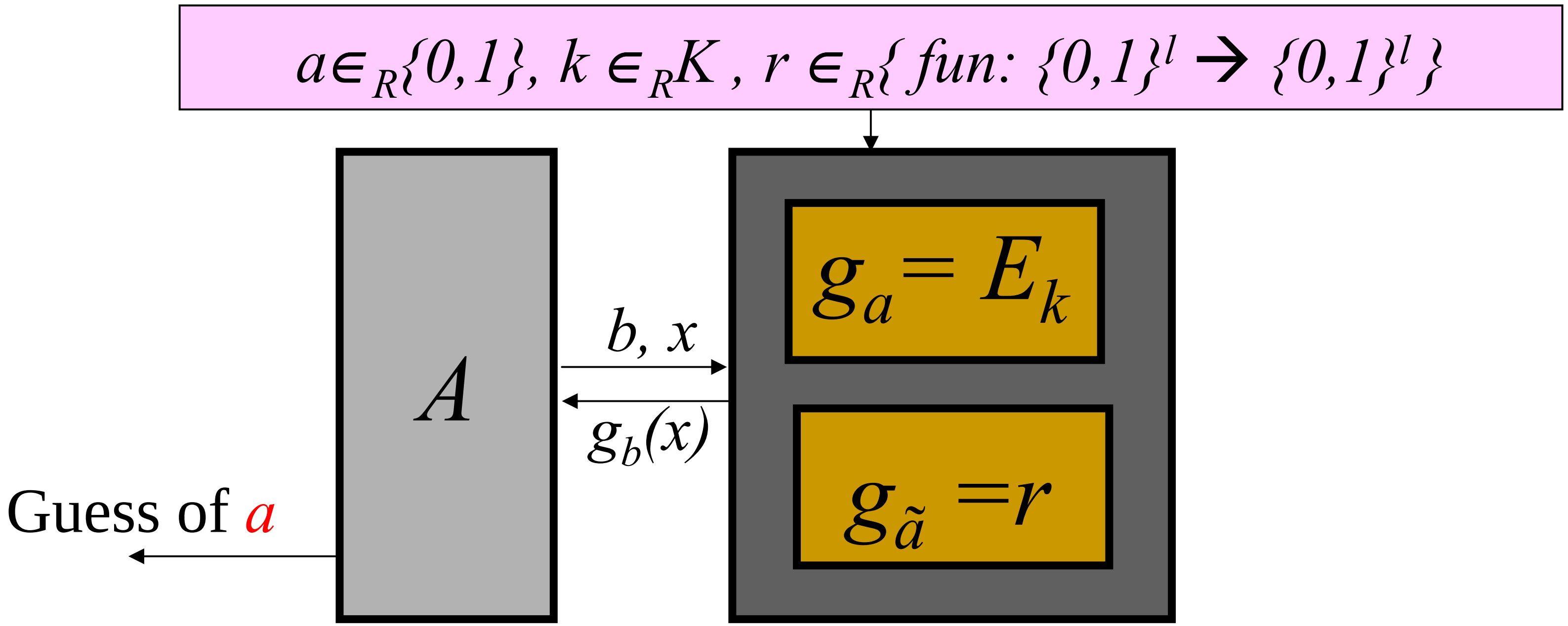


But can we have good *block cipher*?

- Keys must be pretty long
- Blocks must be pretty long
  - Typically 64-128 bits/block
- Knowledge of some plaintext-ciphertext pairs should not expose key or other plaintext-ciphertext pairs
- A random permutation is certainly enough…
  - Think of it as being built dynamically:
    - New input x: select (unused) value for p(x) randomly
    - Repeating input x: return (previously used) p(x)
    - key = identifier of permutation
- Problem: too many random permutations
  - 2^l strings, therefore $2^l!$ permutations over l bits
- Solution?

Pseudo-Random Permutation

- Adversary has oracle access to two black boxes – one containing the PRP, the other a random function…

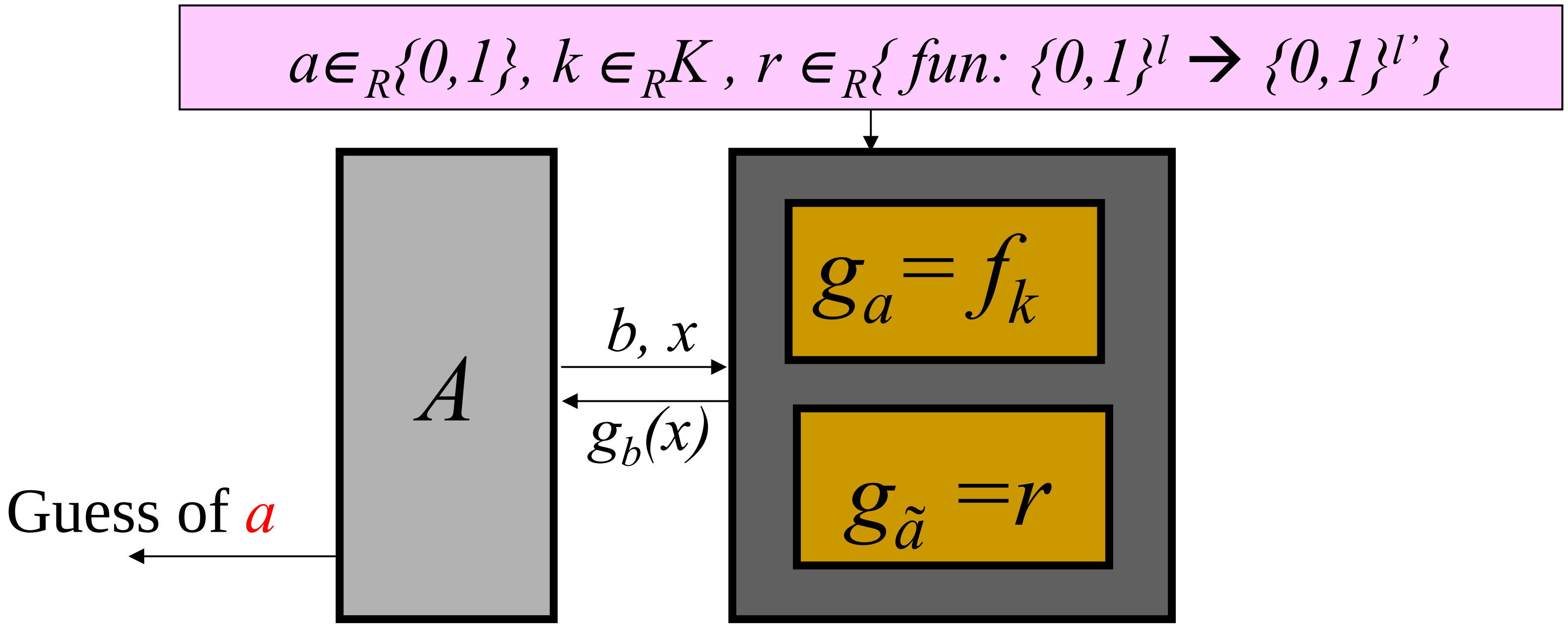


Defining Pseudo-Random Permutation

- Given algorithm A^f with oracle to function $f:\{0,1\}^l \rightarrow \{0,1\}^l$
- Let $ADV^{PRP}_{A,E} = \Pr(A^{E_k} = 1) - \Pr(A^r = 1)$ where $k \in_R K$ and r is a random function $r:\{0,1\}^l \rightarrow \{0,1\}^l$
- Let $ADV^{PRP}_E(t,q)=MAX\{ADV_{A,E}\}$ for A limited to time t and q queries
  - Should be negligible for feasible t, q
  - Ideally: $ADV^{PRP}_E(t,q)=c \cdot 1/(|K|-t)$
- Asymptotically: for every positive polynomials p, T and Q, for \`sufficiently long\` block size l, $ADV^{PRP}_E(t,q)<1/p(l)$ for every $t<T(l)$, $q<Q(l)$.
- Adversary controls plaintext → chosen plaintext attack
  - Exercise: modify definition to allow also chosen ciphertext

Pseudo-Random Function

- Like PRP, except a function, not permutation
- Domain and range may differ
- Constructions: PRP from PRF, PRF from PRP



Using PRPs and PRFs

- Share pseudo random function / permutation
  - By sharing a secret (pseudo-random) key
  - Derive many sub-keys: $k_{auth}=E_k(\text{“auth”})$, $k_{2003}=E_k(2003), \ldots$
  - See such applications later (e.g. in TLS/SSL)
- Generate pseudo-random bits
- Criteria for a good cipher
  - Also: use ciphers when you need PRP
- But how can we confirm a cipher is a PRP?

Confirming Security for Cipher

- Unconditional security: often not feasible / too wasteful
- Conditional security: Block Cipher as PRP
  - Proof of ***reduction*** to a `hard` problem
    - Break scheme → prove P=NP, etc.
    - Break scheme → cryptoanalyze standard design
    - Not practical – wasteful constructions, asymptotic proofs
    - More useful: construct more advanced functions from ciphers
  - Check for known (families of) attacks
    - Allow strong attack models (chosen plaintext/ciphertext, etc.)
    - What about other attacks?
  - Confirm by failure of extensive cryptanalysis efforts
    - Very expensive and time consuming
    - Using public designs (Kerckhoff's idea) helps
    - →done for few standards

Practical Block Ciphers

- DES – Data Encryption Standard
  - 16-round Feistel cipher
  - 64 bits input/output blocks, 56 bits key
    - Vulnerable to exhaustive search – key too short
    - Also: attacks with e.g. $q=2^{40}$ chosen plaintexts
  - Designed in 70's by IBM for NIST
  - Criticized for unpublished design decisions
    - Although actual design is public
  - Efficient hardware implementations; slower software
- Triple DES – used mostly in banking
  - Three applications of DES with two keys: $DES^3_{k1,k2}(m)=E_{k1}(D_{k2}(E_{k1}(m)))$
  - Key: 112 bits; compatible: $DES^3_{k,k}(m)=DES_k(m)$
  - `Double DES` subject to *meet-in-middle* attack

Meet in the Middle Attack

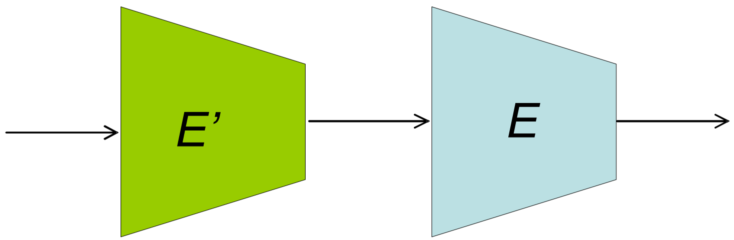
- Goal: \`effective key length\` of 112 bits
- Given *m*, let $c=DES^2_{k,k'}(m)=E_k(E_{k'}(m))$
- For $x'=0^{56}$ to 1^{56}: $y[x']=E_{x'}(m)$
- For $x=0^{56}$ to 1^{56}: $z[x]=D_x(c)$
- Find all <*x,x'*> s.t. $y[x']=z[x]$
  - These are candidate keys ($k=x$, $k'=x'$)
  - At most 2^{56} such pairs (usually less)
  - Test with another plain-ciphertext pair
- Notice: attack works for any cipher/PRP

Practical Block Ciphers - AES

- AES - Advanced Encryption Standard
  - A new NIST standard
  - Selected among 18 proposals submitted to a lengthy, open design and evaluation process
  - Proposal name: Rijndael
  - Goals: improve security and (SW) efficiency (cf. DES)
  - Keys with a length of 128, 192, or 256 bits
    - We hope attack requires almost 2^{128} AES computations
  - Blocks: 128 bits

Cryptanalysis-tolerant Cipher

- Suppose *E, E'* are two candidate ciphers
  - E.g., a standard (AES) and a proprietary
  - Maybe AES will be cryptanalyzed? Maybe our proprietary cipher is easy to cryptanalyze?
- Cascade [EG85]: $E^*=E\circ E'$
- E^* is PRP if <u>either</u> E <u>or</u> E' is PRP
  - We say that cascade is <u>cryptanalysis tolerant</u>



Cascading Ciphers (PRPs)

- Given two PRP candidate functions, *f* and *f'*, define: $h_{k,k'}(x) = f_k(f'_{k'}(x))$
- **Claim**: if either *f* or *f'* is a PRP, then *h* is a PRP.
- Proof sketch: Suppose *Adv* can distinguish *h* from random permutation. Then to distinguish *f*, select *k'* and use *Adv* on $f_k(f'_{k'}(x))$; similarly for *f'*. $\blacksquare$
- Motivating *iterative PRP / Block Cipher design*

Minimal Assumptions Principles

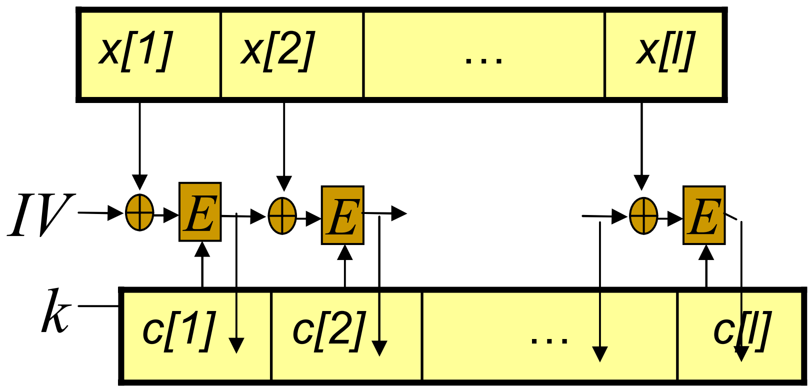
- Perfect (unconditional) security is best – but usually infeasible
- Assumptions should be tested (cryptanalysis)
  - Extensively
  - For specific key and input length
    - Asymptotic analysis helps but not enough for practice
- Use alternative assumptions (cryptanalysis-tolerance), avoid multiple assumptions
- Assumptions should be easy to test
  - Simple, well defined, pessimistic
  - Fixed input length, deterministic functions

Are Block Ciphers Good for Encryption?

- Block ciphers (modeled as PRPs) are easy to test
  - Fixed input length, deterministic functions
- But… `real` plaintext is variable-length!
- Also… what if we encrypt same plaintext?
  - With block ciphers, we get the same ciphertext
  - Sometimes Ok, sometimes – exposure
- Solution: `Modes of Operation` of a cipher
  - Define how to use cipher for encryption
  - Transforming to stream cipher / support VIL
  - Randomized (probabilistic) encryption

Modes of Operation

- Define how to use cipher for encryption
  - *Electronic code book (ECB)* mode: encrypt each plaintext block separately (`trivial` mode)
- Other modes allow…
  - Use cipher as PRG
  - Variable Input Length (VIL)
  - Randomization – hide repeating plaintext
    - Use *Initialization Vector (IV)* – (normally random)
  - Other goals
- *Cipher Block Chaining (CBC)* mode:
- Analysis – in exercise
- First, define encryption…



A shared key encryption scheme…

- Is a triple of algorithms: <*KG, E, D*>
  - *Key Generation, Encryption, Decryption*
- All three: probabilistic, efficient algorithms
  - Asymptotic analysis: efficient=poly time
- For every key *k* and plaintext *p* holds: $p=Dec_k(Enc_k(p))$.

Defining Secure Encryption

- Intuition: *without secret key, Adversary learns nothing useful about the plaintext*
- Questions:
  - What is `useful` new knowledge about plaintext?
  - What is the distribution of the plaintext?
  - Can we be sure of security? Under what assumptions?
- Security as indistinguishability…
  - Let attacker select any two plaintexts
    - Could be very similar… or some special message
  - Select encryption of <u>one</u> of the two
  - Attacker should not be able to find which!
  - *Attack model*: known/chosen plaintext, chosen ciphertext…

Chosen Plaintext Indistinguishability Test

- Given algorithm A^E with oracle to E_k
  - Chosen plaintext attack
- CPA-IND: (CPA Indistinguishability Test)
  - $k \leftarrow KG\,(\,);$
  - $(p[1], p[2], state) \leftarrow A^E(\text{“select inputs”});$
  - $b \in_R \{0,1\};$
  - $b' \leftarrow A^E(\text{“distinguish”}, p[1], p[2], state);$
  - If $b' = b$ return (win) else return (loss);

$$ADV^{CPA-IND}_{A,<KG,E,D>} = \Pr(CPA-IND^{A,<KG,E,D>} = \text{"win"}) - \tfrac{1}{2}$$

CPA-IND Secure Cryptosystem

- Let C=<KG,E,D>
- $ADV^{CPA\text{-}IND}{}_{C}(t,q)=MAX\{ADV_{A,E}\}$ for *A* limited to time t and q queries
  - Should be negligible for feasible t, q
- Asymptotically: for every positive polynomials p, T and Q, for \`sufficiently long\` block size l, $ADV^{CPA\text{-}IND}{}_{C}(t,q)<1/p(l)$ for every $t<T(l)$, $q<Q(l)$.
- Exercise: define for chosen ciphertext attack

Indistinguishability Test is Strong

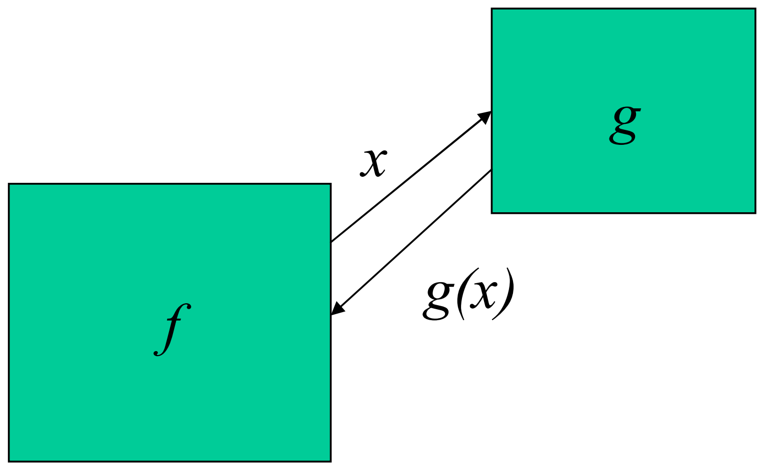
- Two encryptions of the same message should be indistinguishable
  - Otherwise adversary can ask for another encryption of known message and identify it
  - Encryption must be randomized and/or state variable
    - With state variable, encryption depends on history
  - In practice: usually encryption is randomized
- No assumption about the plaintext
  - May be just two messages, ‘0’ and ‘1’
  - May be biased (90% is ‘0’)
- Yet… PRP/PRF → CPA-IND Secure Cryptosystem!

CPA-IND Secure Cryptosystem from PRP

- Let C_k be a block cipher (PRP) or PRF
- Then encrypt each message m using $E_k(m)=r||C_k(m \oplus r)$, where r is random
- Observation: this is simply CBC-mode of C_k with a single block!
  - Proof extends to multiple-block CBC
- Theorem [GM89]: $E_k(m)$ is IND-CPA secure.

In General: Cryptographic Constructions

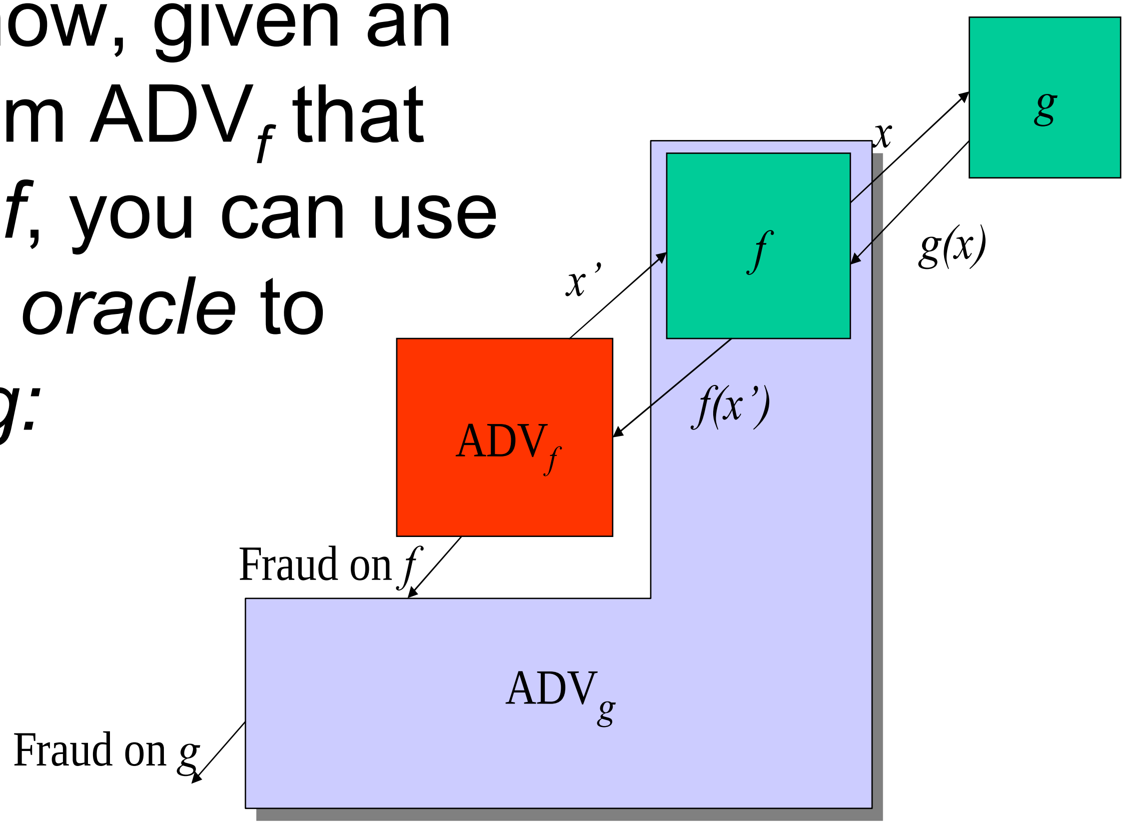
- Build new crypto function *f*, using construction *Π* using function *g*
- Notation: $f = \Pi^g$
- Idea: make *f* for goal *F*, from *g* designed for goal *G*
- Goal *G* is simpler, weaker, easier to test… or we simply have good candidates for G!



Cryptographic Constructions

Proving security

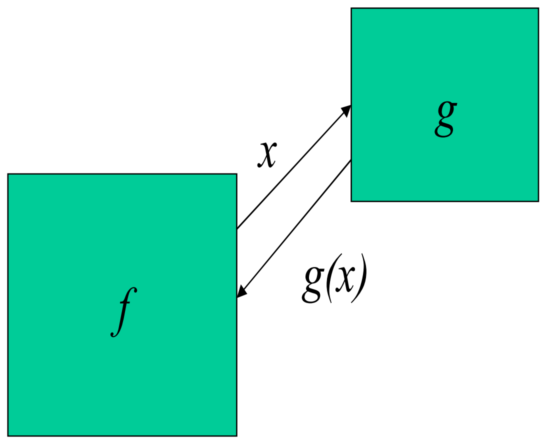
- Show how, given an algorithm ADV_f that breaks *f*, you can use it as an *oracle* to attack *g:*



Cryptographic Constructions

Demonstrating insecurity

- Usual method:
  - Let *g'* be an arbitrary function for goal *G.*
  - Design *g* which also satisfies *G:*
    - Security of *g* follows (easily?) from security of *g'*
    - But *g* is not good for the construction…
    - Namely: the function *f* which is constructed using *g* does *not* satisfy goal *F.*
- Example…



Conclusion: Principles of Cryptography

- Arbitrary Adversary Principle: Assume restrictions on capabilities of adversary – not on adversary's strategy!
- Kerckhoffs' principle: designs are public, only keys are secret
- Sufficient key length Principle:
  - Number of possible keys should be large enough
  - To make attacks infeasible, using best adversary resources expected during `sensitivity period` of data
- Limited key usage principle
- Base security on simple, well-tested assumptions, preferably - allow for failure of some assumption (cryptanalysis-tolerance)