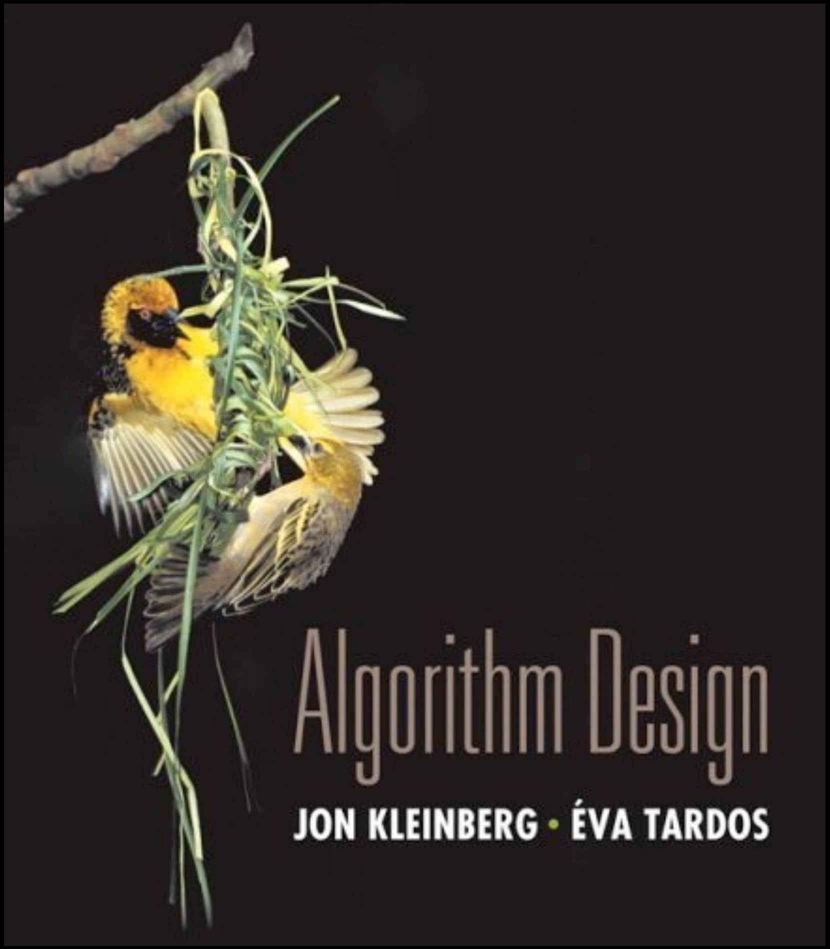


Chapter 13

Randomized Algorithms

Slides by Kevin Wayne.

Randomization

Algorithmic design patterns.

- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

in practice, access to a pseudo-random number generator

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simpler, faster, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.

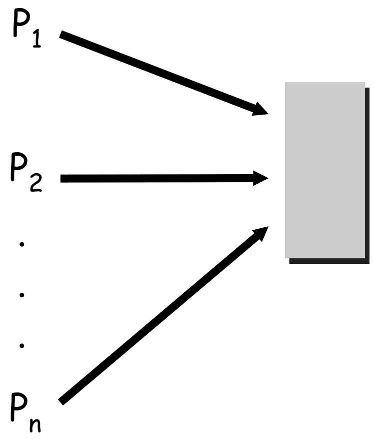
13.1 Contention Resolution

Contention Resolution in a Distributed System

Contention resolution. Given n processes P_1, …, P_n, each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

Restriction. Processes can't communicate.

Challenge. Need symmetry-breaking paradigm.



Contention Resolution: Randomized Protocol

Protocol. Each process requests access to the database at time t with probability p = 1/n.

Claim. Let S[i, t] = event that process i succeeds in accessing the database at time t. Then $1/(e \cdot n) \leq \Pr[S(i, t)] \leq 1/(2n)$.

Pf. By independence, $\Pr[S(i, t)] = p\,(1-p)^{n-1}$.

process i requests access

none of remaining n-1 processes request access

- Setting p = 1/n, we have $\Pr[S(i, t)] = 1/n\ \underbrace{(1 - 1/n)^{n-1}}_{\text{between } 1/e \text{ (limit } n\to\infty) \text{ and } 1/2\ (n=2)}$. ▪

value that maximizes Pr[S(i, t)]

Useful facts from calculus. As n increases from 2, the function:

- $(1 - 1/n)^n$ converges monotonically from 1/4 up to 1/e
- $(1 - 1/n)^{n-1}$ converges monotonically from 1/2 down to 1/e.

Contention Resolution: Randomized Protocol

Claim. The probability that process i fails to access the database in **e • n** rounds is at most **1/e**. After **e·n(c ln n)** rounds, the probability is at most $\mathbf{n^{-c}}$.

Pf. Let F[i, t] = event that process i fails to access database in rounds 1 through t. By independence and previous claim, we have Pr[F(i, t)] $\le$ $(1 - 1/(en))^t$.

- Choose t = $\lceil e \cdot n \rceil$: $\Pr[F(i,t)] \le \left(1 - \frac{1}{en}\right)^{\lceil en \rceil} \le \left(1 - \frac{1}{en}\right)^{en} \le \frac{1}{e}$
- Choose t = $\lceil e \cdot n \rceil \lceil c \ln n \rceil$: $\Pr[F(i,t)] \le \left(\frac{1}{e}\right)^{c \ln n} = n^{-c}$

Contention Resolution: Randomized Protocol

Claim. The probability that all processes succeed within **$2e \cdot n \ln n$** rounds is at least **$1 - 1/n$**.

Pf. Let F[t] = event that at least one of the n processes fails to access database in any of the rounds 1 through t.

$$\Pr[F[t]] = \Pr \bigcup_{i=1}^{n} F[i,t] \underset{\substack{\uparrow \\ \text{union bound}}}{\le} \sum_{i=1}^{n} \Pr[F[i,t]] \underset{\substack{\uparrow \\ \text{previous slide}}}{\le} n\left(1-\tfrac{1}{en}\right)^{t}$$

- Choosing $t = \lceil en \rceil \lceil 2 \ln n \rceil$ yields $\Pr[F[t]] \le n \cdot n^{-2} = 1/n$. ▪

Union bound. Given events $E_1, \ldots, E_n$, $\Pr \bigcup_{i=1}^{n} E_i \le \sum_{i=1}^{n} \Pr[E_i]$

13.2 Global Minimum Cut

Global Minimum Cut

Global min cut. Given a connected, undirected graph **G = (V, E)** find cut **(A, B)** of minimum cardinality (= number of edges connecting **A & B**).

Applications. Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design.

Network flow solution.

- Replace every edge (u, v) with two antiparallel edges (u, v) and (v, u).
- Pick some vertex s and compute min s-v cut separating s from each other vertex $v \in V$.

Resulting False intuition. Global min-cut is harder than min s-t cut.

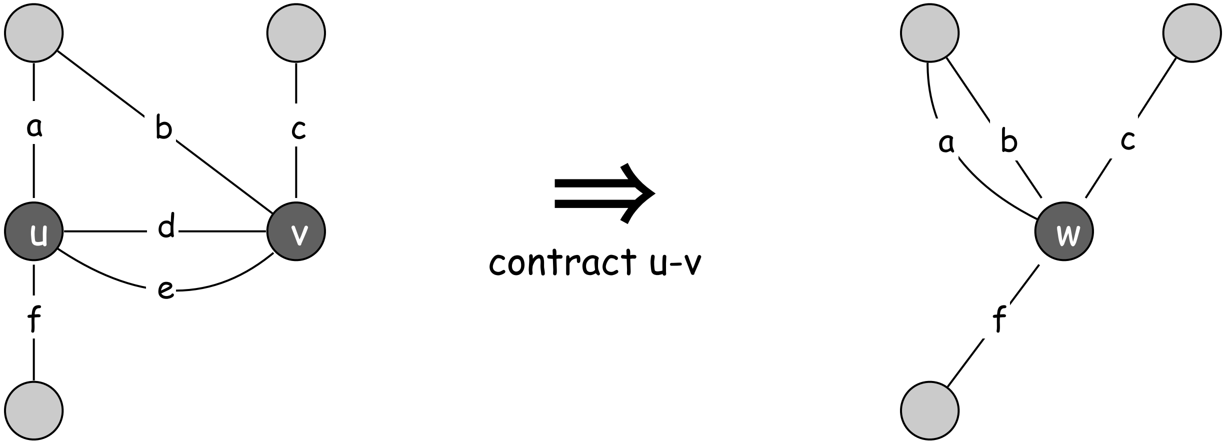
Contraction Algorithm

Contraction algorithm. [Karger 1995]

- Pick an edge e = (u, v) uniformly at random.

- Contract edge e.
  - replace u and v by single new super-node w
  - preserve edges, updating endpoints of u and v to w
  - keep parallel edges, but delete self-loops

- Repeat until graph has just two nodes v_1 and v_2.
- Return the cut (all nodes that were contracted to form v_1).

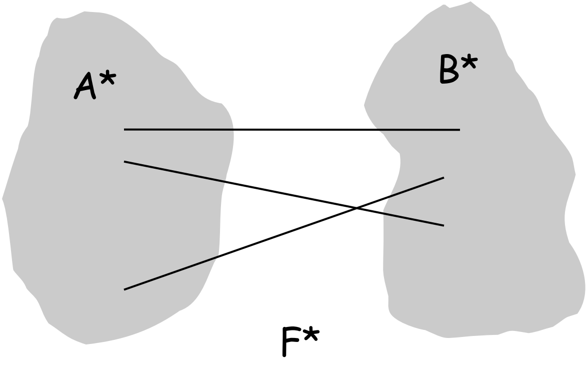


Contraction Algorithm

Claim. The contraction algorithm returns a min cut with **prob** $\geq$ **2/n^2**.

Pf. Consider a global min-cut (A*, B*) of G. Let F* be edges with one endpoint in A* and the other in B*. Let k = |F*| = size of min cut.

- In first step, algorithm contracts an edge in F* with probability k / |E|.
- Every node has degree $\geq$ k since otherwise (A*, B*) would not be min-cut. $\Rightarrow$ $|E| \geq \frac{1}{2}kn$.
- Thus, algorithm contracts an edge in F* with probability $\leq$ 2/n.



Contraction Algorithm

Claim. The contraction algorithm returns a min cut with **prob** $\geq$ **$2/n^2$**.

Pf. Consider a global min-cut (A*, B*) of G. Let F* be edges with one endpoint in A* and the other in B*. Let k = |F*| = size of min cut.

- Let G' be graph after j iterations. There are n' = n-j supernodes.
- Suppose no edge in F* has been contracted. The min-cut in G' is still k.
- Since value of min-cut is k, $|E'| \geq \frac{1}{2}kn'$.
- Thus, algorithm contracts an edge in F* with probability $\leq 2/n'$.

- Let E_j = event that an edge in F* is not contracted in iteration j.

$$\begin{aligned}
\Pr[E_1 \wedge E_2 \cdots \wedge E_{n-2}] &= \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \cdots \times \Pr[E_{n-2} \mid E_1 \wedge E_2 \cdots \wedge E_{n-3}] \\
&\geq \left(1-\tfrac{2}{n}\right)\left(1-\tfrac{2}{n-1}\right)\cdots\left(1-\tfrac{2}{4}\right)\left(1-\tfrac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&\geq \frac{2}{n^2}
\end{aligned}$$

Contraction Algorithm

Amplification. To amplify the probability of success, run the contraction algorithm many times.

Claim. If we repeat the contraction algorithm **n^2 ln n** times with independent random choices, the probability of failing to find the global min-cut is at most **$1/n^2$**.

Pf. By independence, the probability of failure is at most

$$\left(1-\frac{2}{n^2}\right)^{n^2\ln n} = \left(\left(1-\frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right)^{2\ln n} \underset{\substack{\big\uparrow \\ (1-1/x)^x \le 1/e}}{\le} \left(e^{-1}\right)^{2\ln n} = \frac{1}{n^2}$$

Global Min Cut: Context

Remark. Overall running time is slow since we perform $\Theta(\mathbf{n^2 \log n})$ iterations and each takes $\mathbf{\Omega(m)}$ time.

Improvement. [Karger-Stein 1996] $\mathbf{O(n^2 \log^3 n)}$.

- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits **50%** when $\mathbf{n/\sqrt{2}}$ nodes remain.
- Run contraction algorithm until $\mathbf{n/\sqrt{2}}$ nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.

Extensions. Naturally generalizes to handle positive weights.

Best known. [Karger 2000] $\mathbf{O(m \log^3 n)}$.

faster than best known max flow algorithm or deterministic global min cut algorithm

13.3 Linearity of Expectation

Expectation

Expectation. Given a discrete random variables X, its expectation E[X] is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \Pr[X = j]$$

Waiting for a first success. Coin is heads with probability p and tails with probability 1-p. How many independent flips X until first heads?

$$E[X] \;=\; \sum_{j=0}^{\infty} j \cdot \Pr[X = j] \;=\; \sum_{j=0}^{\infty} j \, \underset{\substack{\uparrow \\ \text{j-1 tails}}}{(1-p)^{j-1}} \, \underset{\substack{\uparrow \\ \text{1 head}}}{p} \;=\; \frac{p}{1-p} \sum_{j=0}^{\infty} j \, (1-p)^{j} \;=\; \frac{p}{1-p} \cdot \frac{1-p}{p^2} \;=\; \frac{1}{p}$$

Expectation: Two Properties

Useful property. If X is a 0/1 random variable, E[X] = Pr[X = 1].

Pf.

$$E[X] \;=\; \sum_{j=0}^{\infty} j \cdot \Pr[X = j] \;=\; \sum_{j=0}^{1} j \cdot \Pr[X = j] \;=\; \Pr[X = 1]$$

not necessarily independent

Linearity of expectation. Given two random variables X and Y defined over the same probability space, E[X + Y] = E[X] + E[Y].

Decouples a complex calculation into simpler pieces.

Guessing Cards

Game. Shuffle a deck of n cards; turn them over one at a time; try to guess each card.

Memoryless guessing. No psychic abilities; can't even remember what's been turned over already. Guess a card from full deck uniformly at random.

Claim. The expected number of correct guesses is 1.

Pf. (surprisingly effortless using linearity of expectation)

- Let X_i = 1 if i^{th} prediction is correct and 0 otherwise.
- Let X = number of correct guesses = $X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1/n$.
- $E[X] \underset{\uparrow}{=} E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1.$ ▪

linearity of expectation

Guessing Cards

Game. Shuffle a deck of n cards; turn them over one at a time; try to guess each card.

Guessing with memory. Guess a card uniformly at random from cards not yet seen.

Claim. The expected number of correct guesses is $\Theta(\log n)$.

Pf.

- Let X_i = 1 if i^{th} prediction is correct and 0 otherwise.
- Let X = number of correct guesses = $X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - i - 1)$.
- $E[X] \underset{\uparrow}{=} E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/2 + 1/1 = \underset{\uparrow}{H(n)}$. ▪

linearity of expectation

$\ln(n+1) < H(n) < 1 + \ln n$

Coupon Collector

Coupon collector. Each box of cereal contains a coupon. There are n different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have ≥ 1 coupon of each type?

Claim. The expected number of steps is Θ(**n log n**).

Pf.

- Phase j = time between j and j+1 distinct coupons.
- Let X_j = number of steps you spend in phase j.
- Let X = number of steps in total = $X_0 + X_1 + \ldots + X_{n-1}$.

$$E[X] \;=\; \sum_{j=0}^{n-1} E[X_j] \;\underset{\big\uparrow}{=}\; \sum_{j=0}^{n-1} \frac{n}{n-j} \;=\; n \sum_{i=1}^{n} \frac{1}{i} \;=\; n\,H(n)$$

prob of success = (n-j)/n
⇒ expected waiting time = n/(n-j)

13.5 Randomized Divide-and-Conquer

Quicksort

Sorting. Given a set of n distinct elements S, rearrange them in ascending order.

```
RandomizedQuicksort(S) {
   if |S| = 0 return

   choose a splitter $a_i$ ∈ S uniformly at random
   foreach (a ∈ S) {
      if      (a < $a_i$) put a in $S^-$
      else if (a > $a_i$) put a in $S^+$
   }
   RandomizedQuicksort($S^-$)
   output $a_i$
   RandomizedQuicksort($S^+$)
}
```

Remark. Can implement in-place.

O(log n) extra space

Quicksort

Running time.

- [Best case.] Select the median element as the splitter: quicksort makes $\Theta(n \log n)$ comparisons.
- [Worst case.] Select the smallest element as the splitter: quicksort makes $\Theta(n^2)$ comparisons.

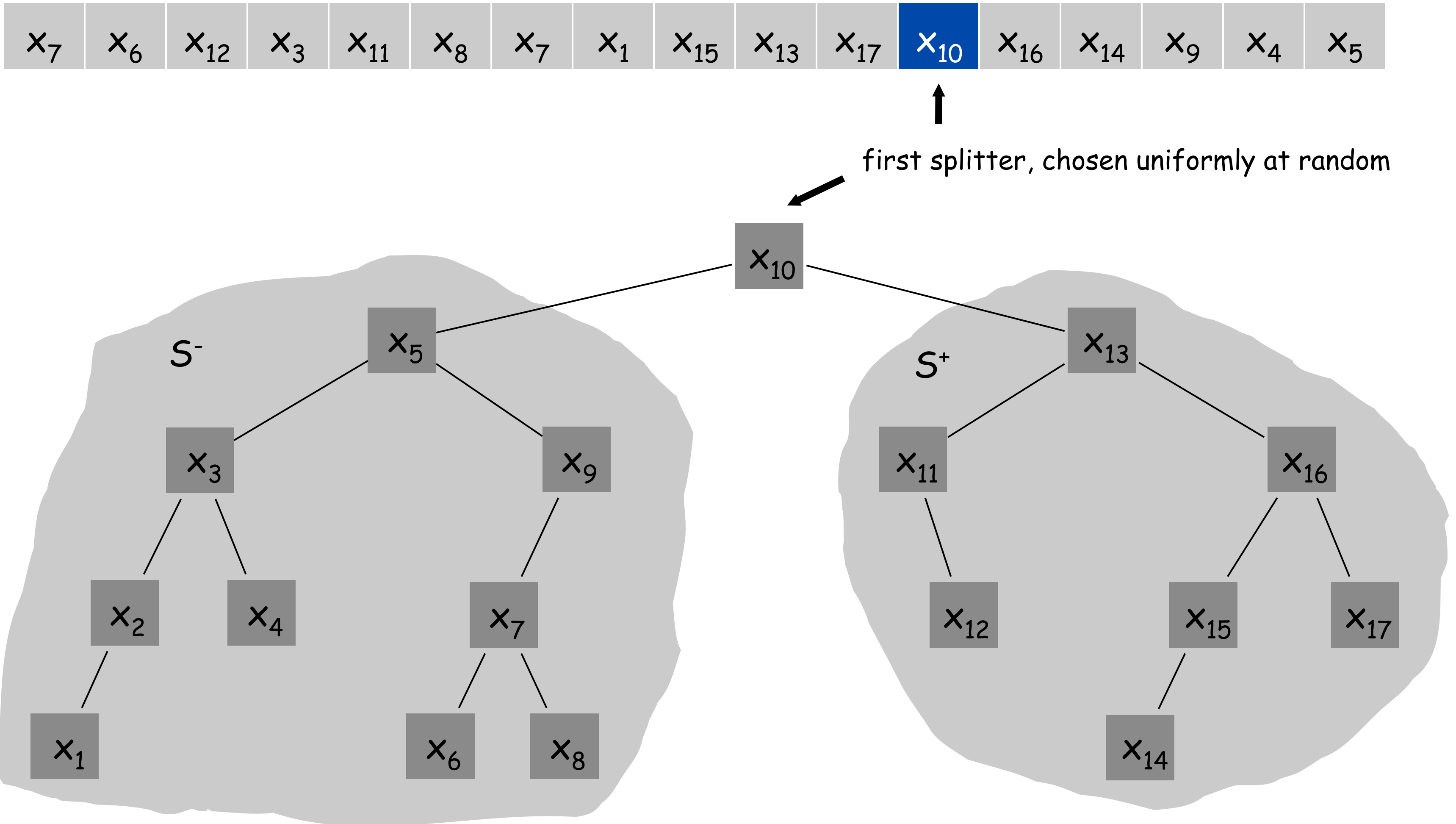
Randomize. Protect against worst case by choosing splitter at random.

Intuition. If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes $\Theta(n \log n)$ comparisons.

Notation. Label elements so that $x_1 < x_2 < \ldots < x_n$.

Quicksort: BST Representation of Splitters

BST representation. Draw recursive BST of splitters.



x_7 x_6 x_{12} x_3 x_{11} x_8 x_7 x_1 x_{15} x_{13} x_{17} x_{10} x_{16} x_{14} x_9 x_4 x_5
x_{10}
S^-
x_5
x_3
x_9
x_2
x_4
x_7
x_1
x_6
x_8
S^+
x_{13}
x_{11}
x_{16}
x_{12}
x_{15}
x_{17}
x_{14}

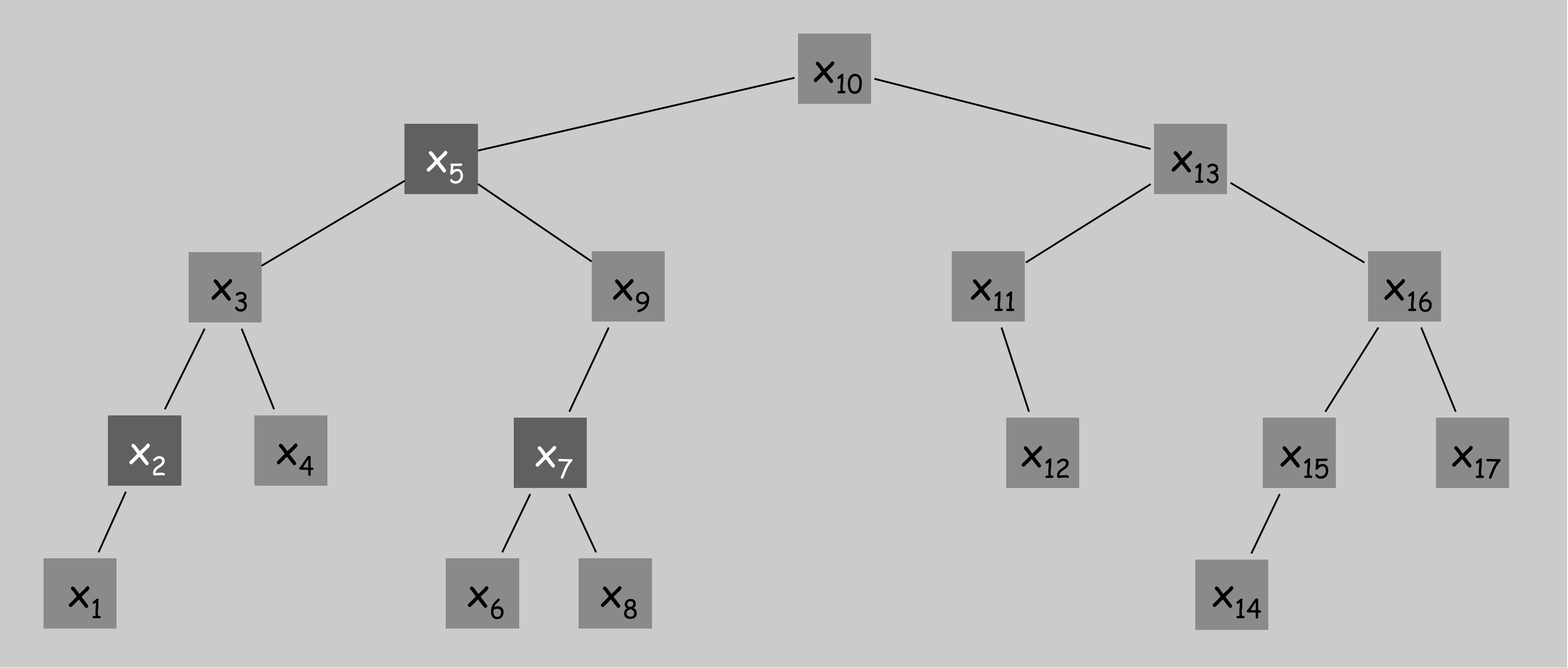
first splitter, chosen uniformly at random

Quicksort: BST Representation of Splitters

Observation. Element only compared with its ancestors and descendants.

- x_2 and x_7 are compared if their lca = x_2 or x_7.
- x_2 and x_7 are not compared if their lca = x_3 or x_4 or x_5 or x_6.

Claim. Pr[x_i and x_j are compared] = 2 / (j - i + 1).
Let C_{ij} be the indicator Rand.Var. of the event "x_i and x_j are compared".



Quicksort: Expected Number of Comparisons

Theorem. Expected # of comparisons is O(n log n).

Pf.

Let C be the Rand.Var. of the # of comparisons.

$$E[C] = E[C_{12}] + E[C_{13}] + E[C_{23}] + \ldots + E[C_{n-2,n-1}] =$$

$$\sum_{1 \le i < j \le n} \frac{2}{j - i + 1} = 2 \sum_{i=1}^{n} \sum_{j=2}^{i} \frac{1}{j} \le 2n \sum_{j=1}^{n} \frac{1}{j} \le 2n \int_{x=1}^{n} \frac{1}{x} dx = 2n \ln n$$

↑

probability that i and j are compared

Theorem. [Knuth 1973] Stddev of number of comparisons is ~ 0.65n.

Ex. If n = 1 million, the probability that randomized quicksort takes less than 4n ln n comparisons is at least 99.94%.

Chebyshev's inequality. $\Pr[|X - \mu| \ge k\delta] \le 1 / k^2$.

Quicksort: Expected Number of Comparisons

The expected number of comparisons in a randomized Quicksort of n elements is (γ is Euler's constant near 0.577) :

$$q_n = 2n \ln n - (4 - 2\gamma)n + 2 \ln n + O(1).$$

In 1996, McDiarmid and Hayward have formulated an exact expression for the probability that the number of comparisons $\boldsymbol{Q_n}$ be far from its average $\boldsymbol{q_n}$

$$\Pr\left[\left|\frac{Q_n}{q_n} - 1\right| > \varepsilon\right] = n^{-(2+o(1))\varepsilon \ln^{(2)} n}$$

Let c be a positive constant. McDiarmid and Hayward's formula imply that there exists another positive constant a smaller than 1 such that

$$\Pr[\, \boldsymbol{Q_n} \in \Theta(n^{1+c})\,] < a^{n^c}.$$

13.6 Universal Hashing

Dictionary Data Type

Dictionary. Given a universe U of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in S is efficient.

Dictionary interface.

- `Create()`: Initialize a dictionary with $S = \emptyset$.
- `Insert(u)`: Add element $u \in U$ to S.
- `Delete(u)`: Delete u from S, if u is currently in S.
- `Lookup(u)`: Determine whether u is in S.

Challenge. Universe U can be extremely large so defining an array of size |U| is infeasible.

Applications. File systems, databases, Google, compilers, checksums P2P networks, associative arrays, cryptography, web caching, etc.

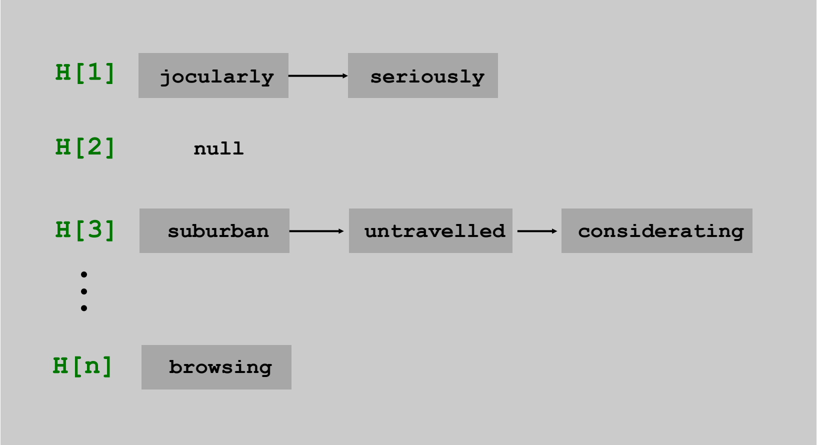
Hashing

Hash function. h : U → { 0, 1, …, n-1 }.

Hashing. Create an array H of size n. When processing element u, access array element H[h(u)].

Collision. When h(u) = h(v) but u ≠ v.

- A collision is expected after Θ(√n) random insertions. This phenomenon is known as the "birthday paradox."
- Separate chaining: H[i] stores linked list of elements u with h(u) = i.



Ad Hoc Hash Function

Ad hoc hash function.

```
int h(String s, int n) {
   int hash = 0;
   for (int i = 0; i < s.length(); i++)
      hash = (31 * hash % n) + s[i];
   return hash % n;
}
```

hash function à la Java string library

Deterministic hashing. If $|U| \geq n^2$, then for any fixed hash function h, there is a subset $S \subseteq U$ of n elements that all hash to same slot. Thus, $\Theta(n)$ time per search in worst-case.

Q. But isn't ad hoc hash function good enough in practice?

Algorithmic Complexity Attacks

When can't we live with ad hoc hash function?

- Obvious situations: aircraft control, nuclear reactors.
- Surprising situations: denial-of-service attacks.

malicious adversary learns your ad hoc hash function (e.g., by reading Java API) and causes a big pile-up in a single slot that grinds performance to a halt

Real world exploits. [Crosby-Wallach 2003]

- Bro server: send carefully chosen packets to D.O.S. the server, using less bandwidth than a dial-up modem
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.

Hashing Performance

Idealistic hash function. Maps m elements uniformly at random to n hash slots.

- Running time depends on length of chains.
- Average length of chain = α = m / n.
- Choose n ≈ m ⇒ on average O(1) per insert, lookup, or delete.

Challenge. Achieve idealized randomized guarantees, but with a hash function where you can easily find items where you put them.

Approach. Use randomization in the choice of h.

↑

adversary knows the randomized algorithm you're using,
but doesn't know random choices that the algorithm makes

Universal Hashing

Universal class of hash functions. [Carter-Wegman 1980s]

- For any pair of elements u≠v ∈ U, $\Pr_{h \in H}[\, h(u) = h(v) \,] \leq 1/n$
- Can select random h efficiently.
- Can compute h(u) efficiently.

chosen uniformly at random

Ex. U = { a, b, c, d, e, f }, n = 2.

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1

$H = \{h_1, h_2\}$

$\Pr_{h \in H}[h(a) = h(b)] = 1/2$

$\Pr_{h \in H}[h(a) = h(c)] = 1$

$\Pr_{h \in H}[h(a) = h(d)] = 0$

. . .

not universal

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1
$h_3(x)$	0	0	1	0	1	1
$h_4(x)$	1	0	0	1	1	0

$H = \{h_1, h_2, h_3, h_4\}$

$\Pr_{h \in H}[h(a) = h(b)] = 1/2$

$\Pr_{h \in H}[h(a) = h(c)] = 1/2$

$\Pr_{h \in H}[h(a) = h(d)] = 1/2$

$\Pr_{h \in H}[h(a) = h(e)] = 1/2$

$\Pr_{h \in H}[h(a) = h(f)] = 0$

. . .

universal

Universal Hashing

Universal hashing property. Let H be a universal class of hash functions; let h ∈ H be chosen uniformly at random from H; and let u ∈ U. For any subset S ⊆ U of size at most n, the expected number of items in S that collide with u is at most 1.

Pf. For any element s ∈ S, define indicator random variable X_s = 1 if h(s) = h(u) and 0 otherwise. Let X be a random variable counting the total number of collisions with u.

$$E_{h \in H}[X] \;=\; E[\textstyle\sum_{s \in S} X_s] \;\underset{\substack{\uparrow \\ \text{linearity of expectation}}}{=}\; \textstyle\sum_{s \in S} E[X_s] \;\underset{\substack{\uparrow \\ X_s \text{ is a 0-1 random variable}}}{=}\; \textstyle\sum_{s \in S} \Pr[X_s = 1] \;\underset{\substack{\uparrow \\ \text{universal} \\ (\text{assumes } u \notin S)}}{\le}\; \textstyle\sum_{s \in S} \frac{1}{n} \;=\; |S| \frac{1}{n} \;\le\; 1$$

Designing a Universal Family of Hash Functions

Theorem. [Bertrand-Chebyshev (1845|1850)]
There exists a prime between n and 2n.

Modulus. Choose a prime number p ≈ n. ← no need for randomness here

Integer encoding. Identify each element $u \in U$ with a base-p integer of r digits: $x = (x_1, x_2, \ldots, x_r)$.

Hash function. Let A = set of all r-digit, base-p integers. For each $a = (a_1, a_2, \ldots, a_r)$ where $0 \le a_i < p$, define

$$h_a(x) = \sum_{i=1}^{r} a_i x_i \mod p$$

Hash function family. $H = \{ h_a : a \in A \}$.

Designing a Universal Class of Hash Functions

Theorem. $H = \{ h_a : a \in A \}$ is a universal class of hash functions.

Pf. Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. We need to show that $\Pr[h_a(x) = h_a(y)] \le 1/n$.

- Since $x \ne y$, there exists an integer j such that $x_j \ne y_j$.
- We have $h_a(x) = h_a(y)$ iff

$$a_j \underbrace{(y_j - x_j)}_{z} = \underbrace{\sum_{i \ne j} a_i (x_i - y_i)}_{m} \mod p$$

- Can assume a was chosen uniformly at random by first selecting all coordinates a_i where $i \ne j$, then selecting a_j at random. Thus, we can assume a_i is fixed for all coordinates $i \ne j$.
- Since p is prime, $a_j z = m \mod p$ has at most one solution among p possibilities. ← see lemma on next slide
- Thus $\Pr[h_a(x) = h_a(y)] = 1/p \le 1/n$. ▪

Number Theory Facts

Fact. Let p be prime, and let $z \neq 0 \bmod p$. Then $\alpha z = m \bmod p$ has at most one solution $0 \leq \alpha < p$.

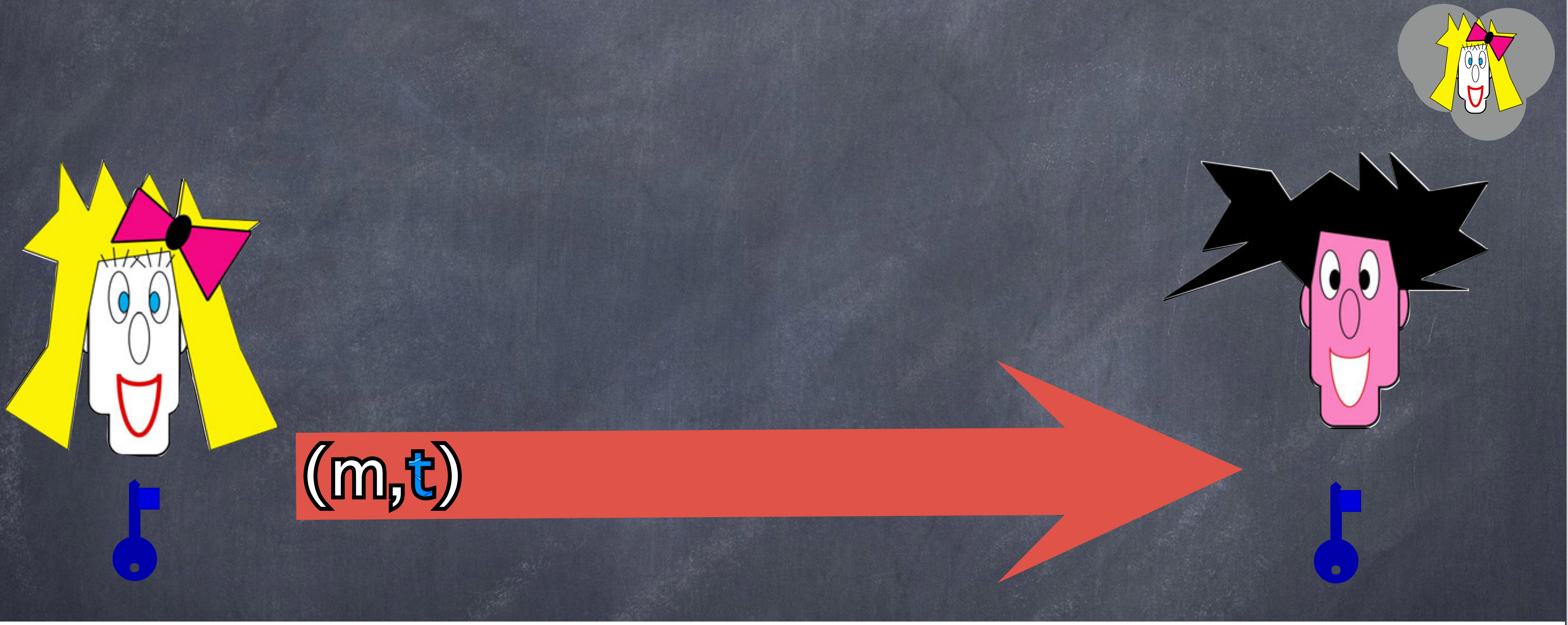
Pf.

- Suppose α and β are two different solutions.
- Then $(\alpha - \beta)z = 0 \bmod p$; hence $(\alpha - \beta)z$ is divisible by p.
- Since $z \neq 0 \bmod p$, we know that z is not divisible by p; it follows that $(\alpha - \beta)$ is divisible by p.
- This implies $\alpha = \beta$. ▪

Bonus fact. Can replace "at most one" with "exactly one" in above fact.
Pf idea. Extended Euclid's algorithm.

Authentication

Symmetric Authentication



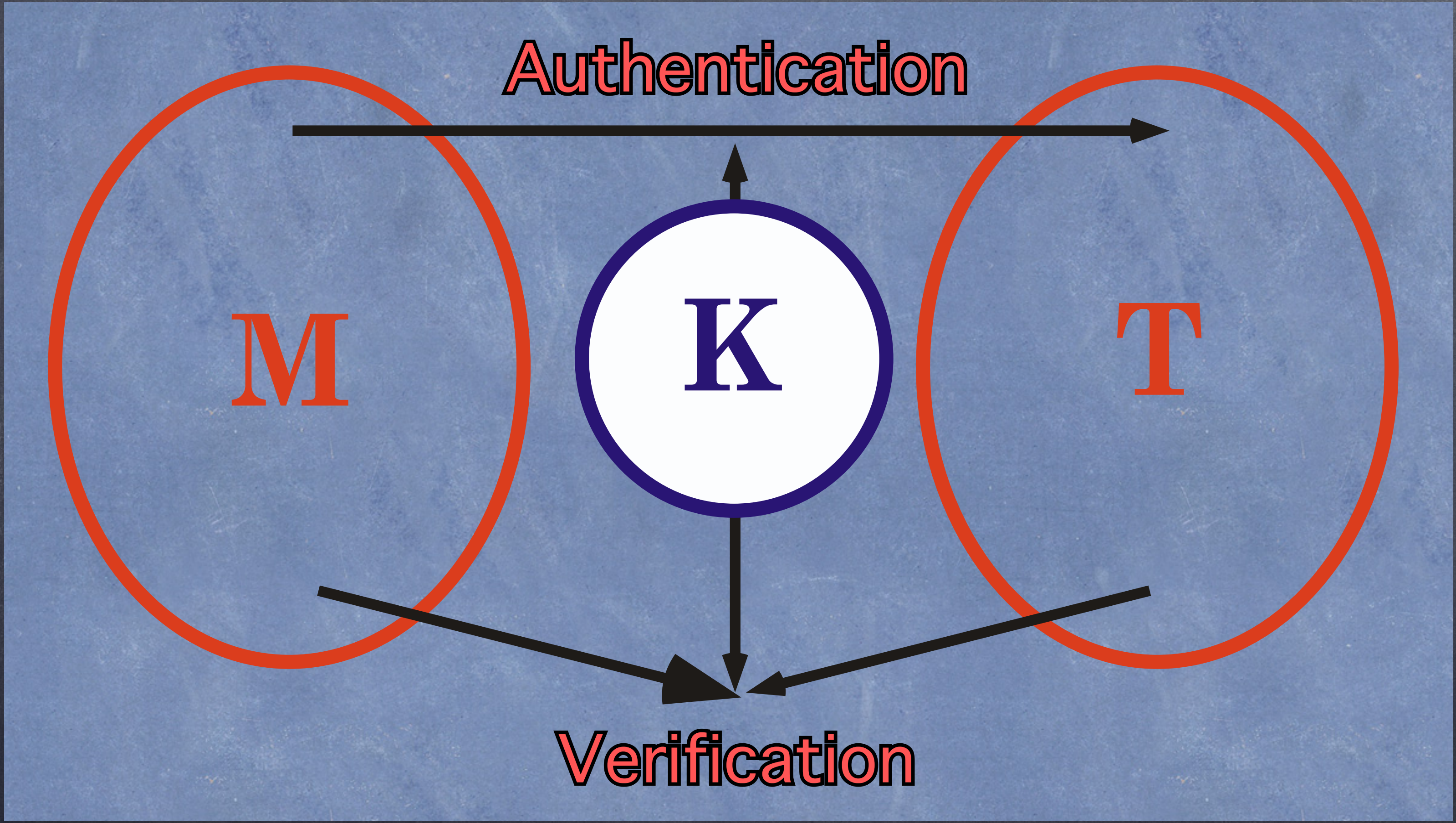
Authentication

t := A🔑(m)

Verification

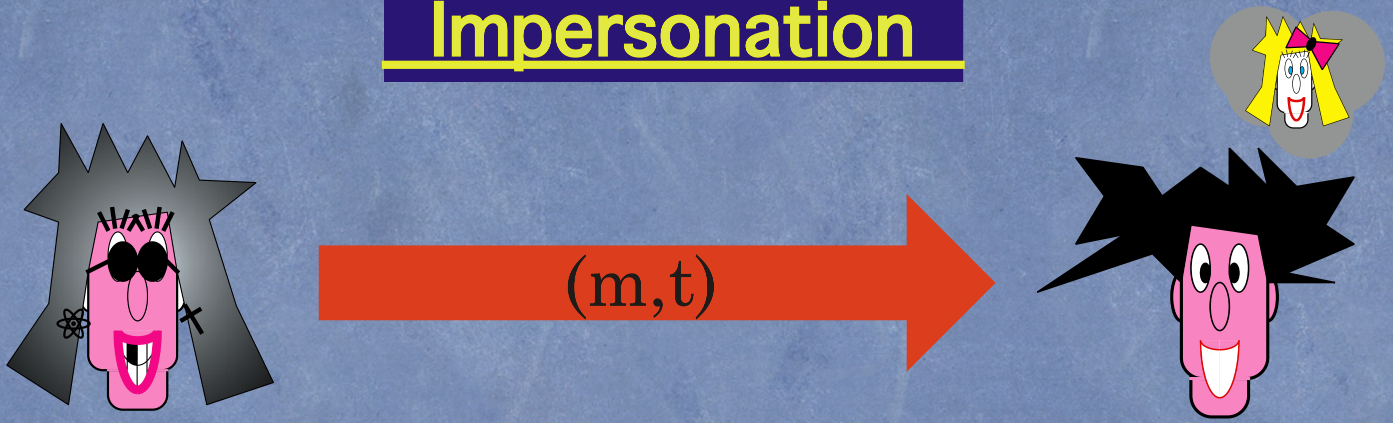
t = A🔑(m) ?

Symmetric Authentication

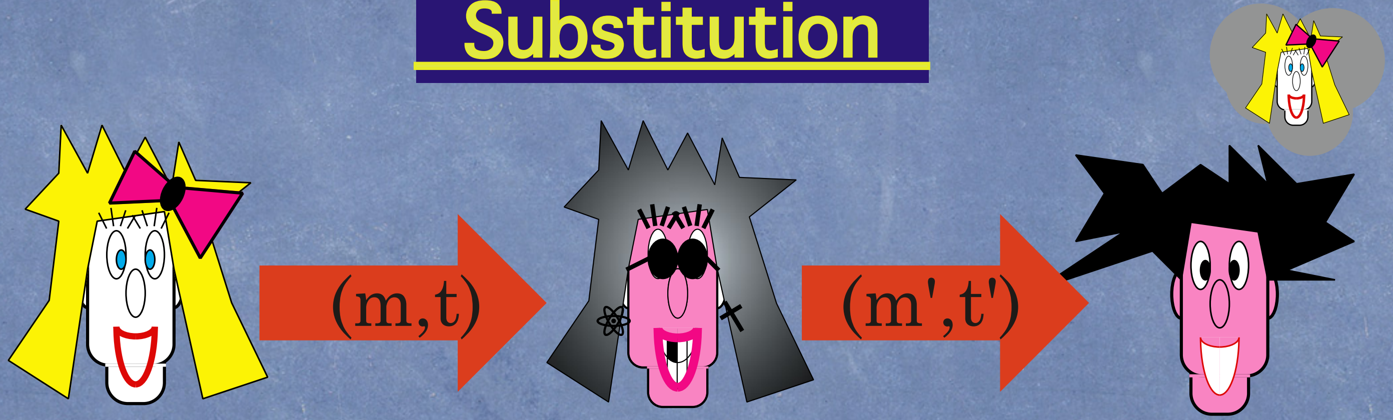


Information Theoretical Security

Impersonation

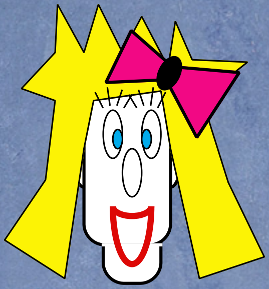


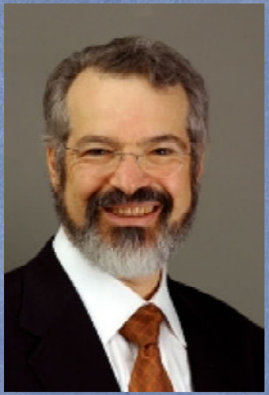
Substitution

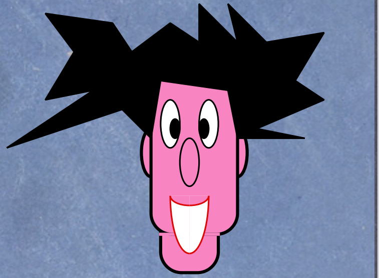


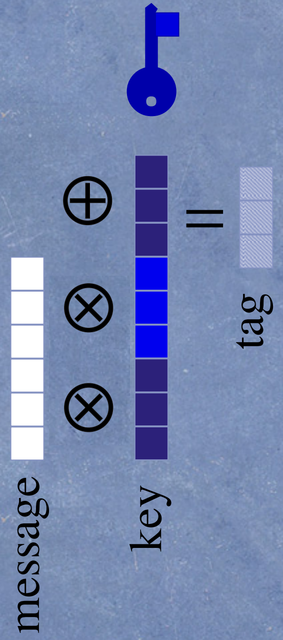
Information Theoretical Security

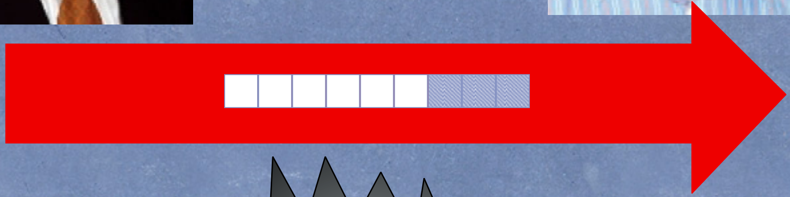
Wegman-Carter One-Time Authentication

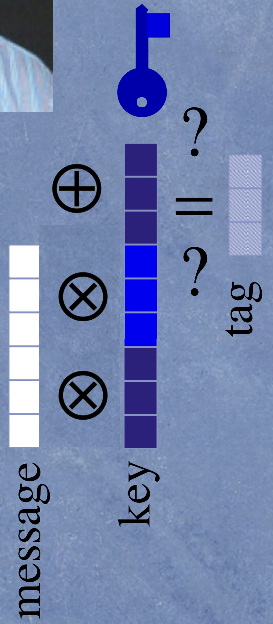


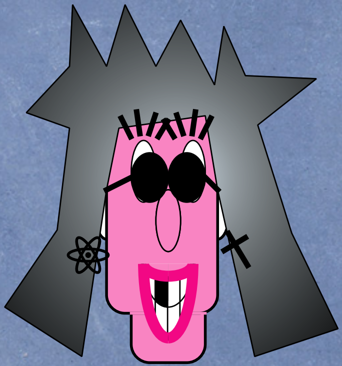






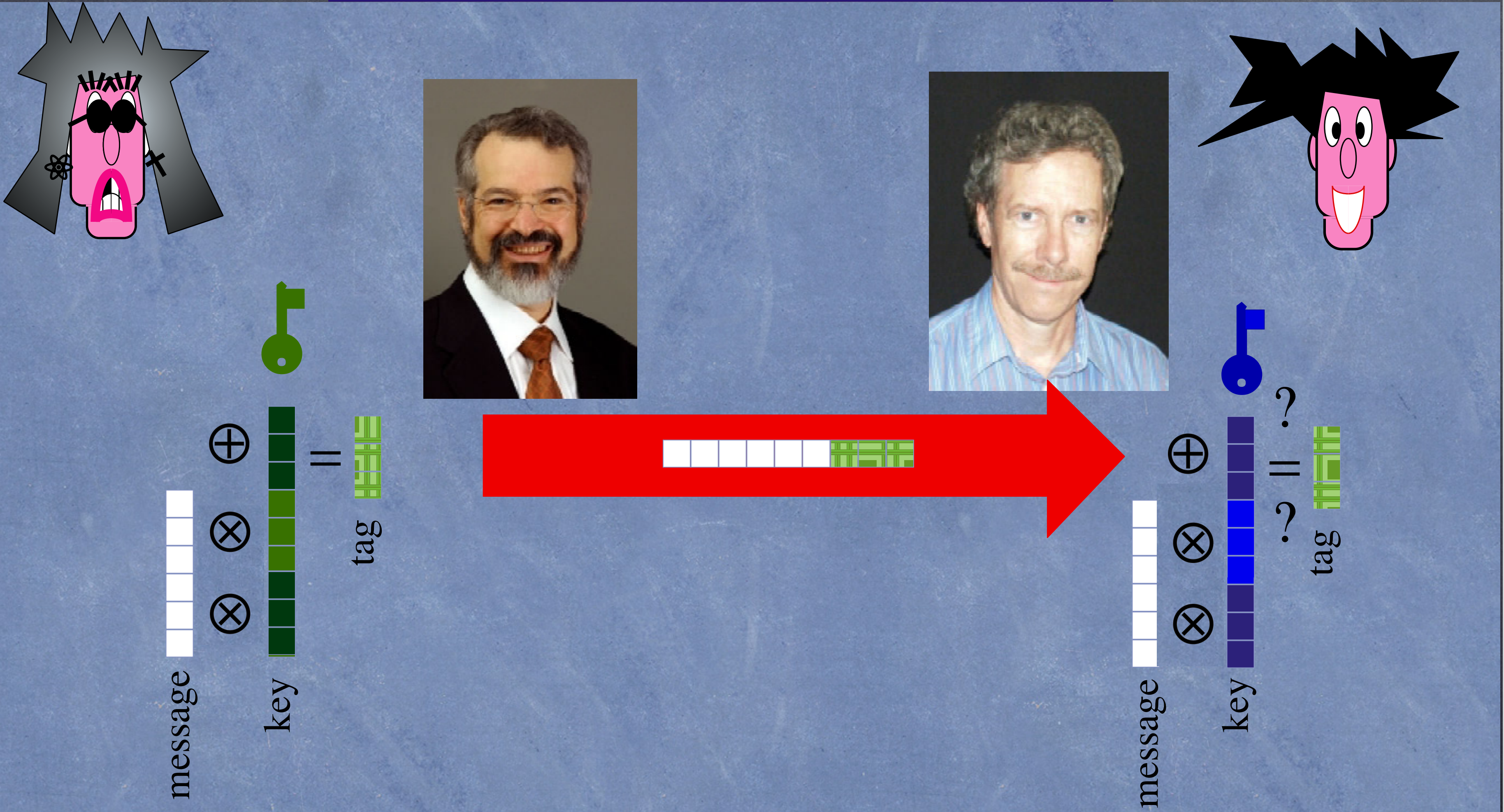






$$h_{a,b}(m)=(\sum_{j=1}^{n} a_j\, m_j+b) \bmod p$$

Wegman-Carter One-Time Authentication



$$m \neq m' \implies \Pr[h_{a,b}(m)=h_{a,b}(m')]=1/p$$

Gemmel-Naor
One-Time Authentication

Authentication

$$t := A_{\text{🔑}}(m)$$

$$|\text{🔑}| \approx 5|t| + \log(|m|)$$

Monte Carlo vs. Las Vegas Algorithms

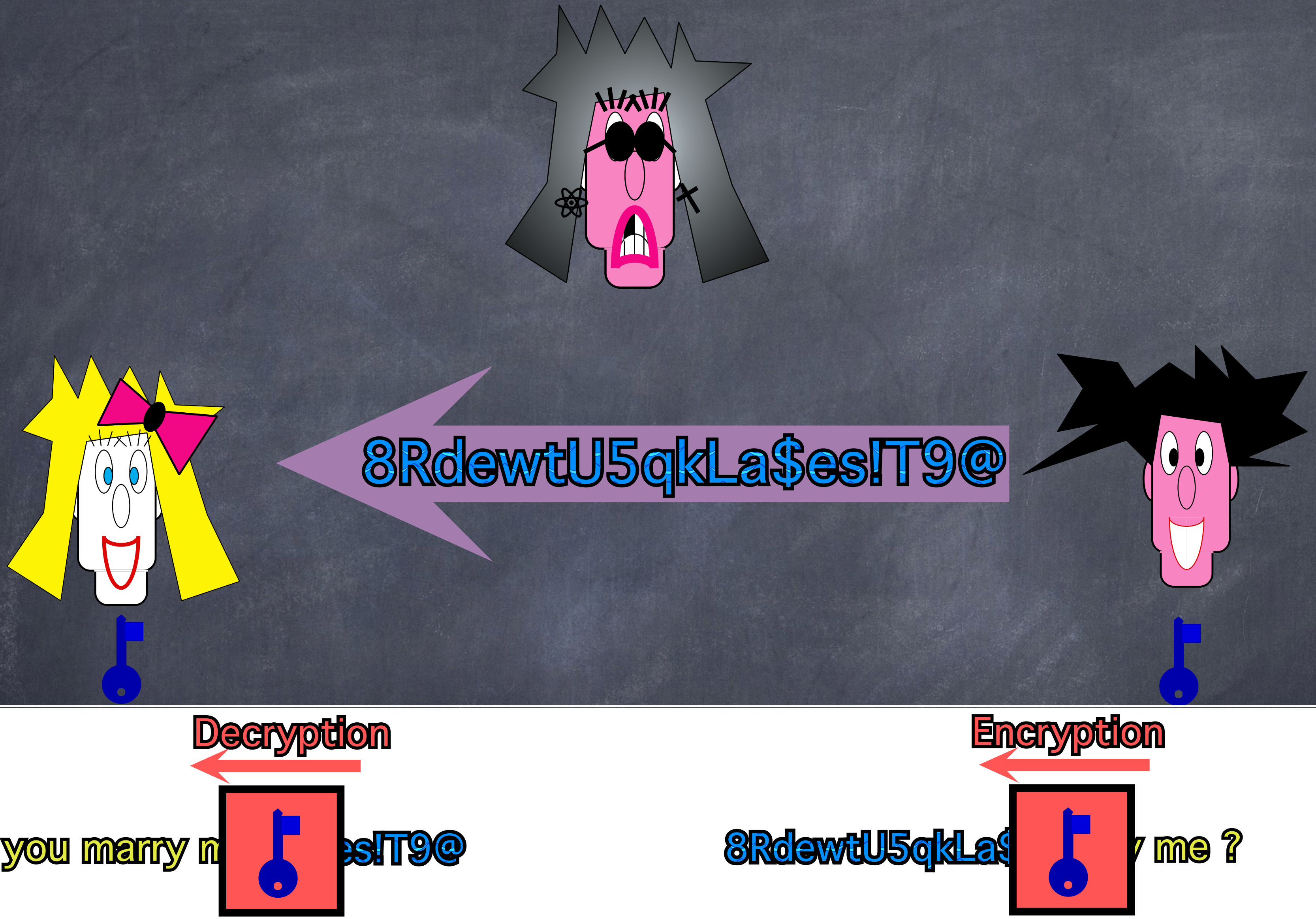
Monte Carlo algorithm. Guaranteed to run in poly-time, likely to find correct answer.
Ex: Contraction algorithm for global min cut.

Las Vegas algorithm. Guaranteed to find correct answer, likely to run in poly-time.
Ex: Randomized quicksort.

stop algorithm after a certain point

Remark. Can always convert a Las Vegas algorithm into Monte Carlo, but no known method to convert the other way.

Encryption

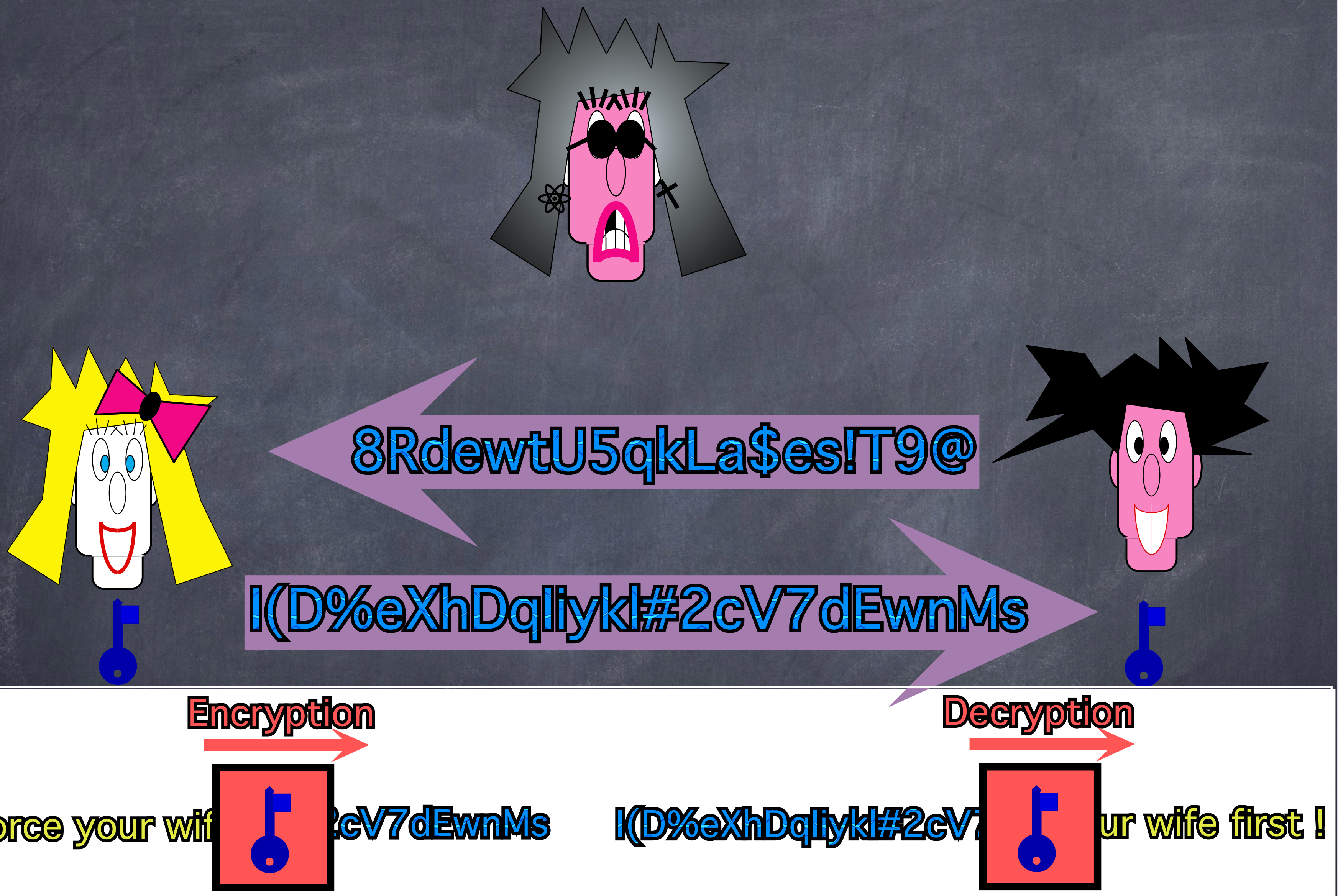


Decryption

Will you marry m es!T9@

Encryption

8RdewtU5qkLa$ y me ?



8RdewtU5qkLa$es!T9@

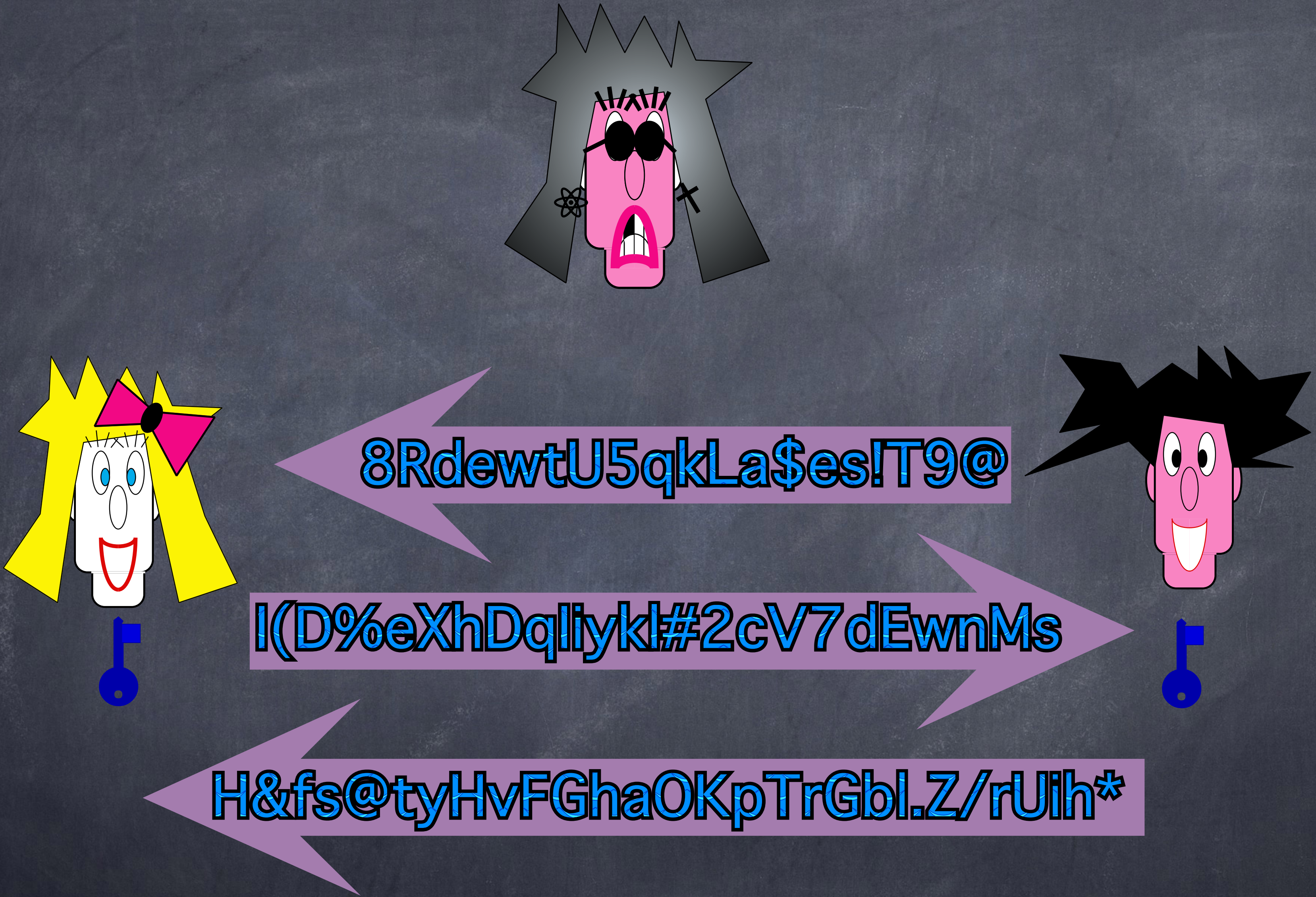
I(D%eXhDqliykl#2cV7dEwnMs

Encryption

Decryption

Divorce your wif 2cV7dEwnMs

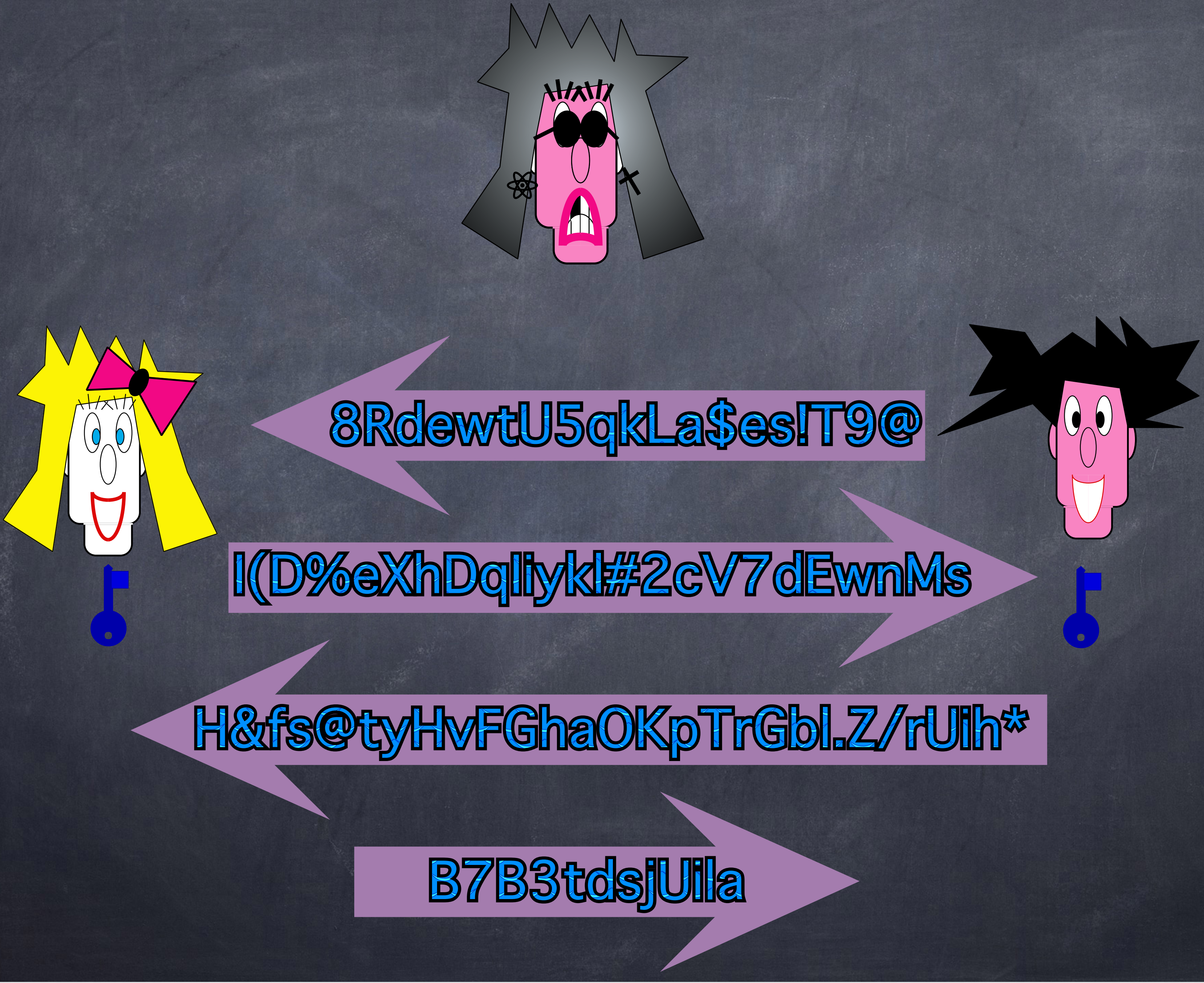
I(D%eXhDqliykl#2cV7 ur wife first !



8RdewtU5qkLa$es!T9@

I(D%eXhDqliykl#2cV7dEwnMs

H&fs@tyHvFGhaOKpTrGbl.Z/rUih*



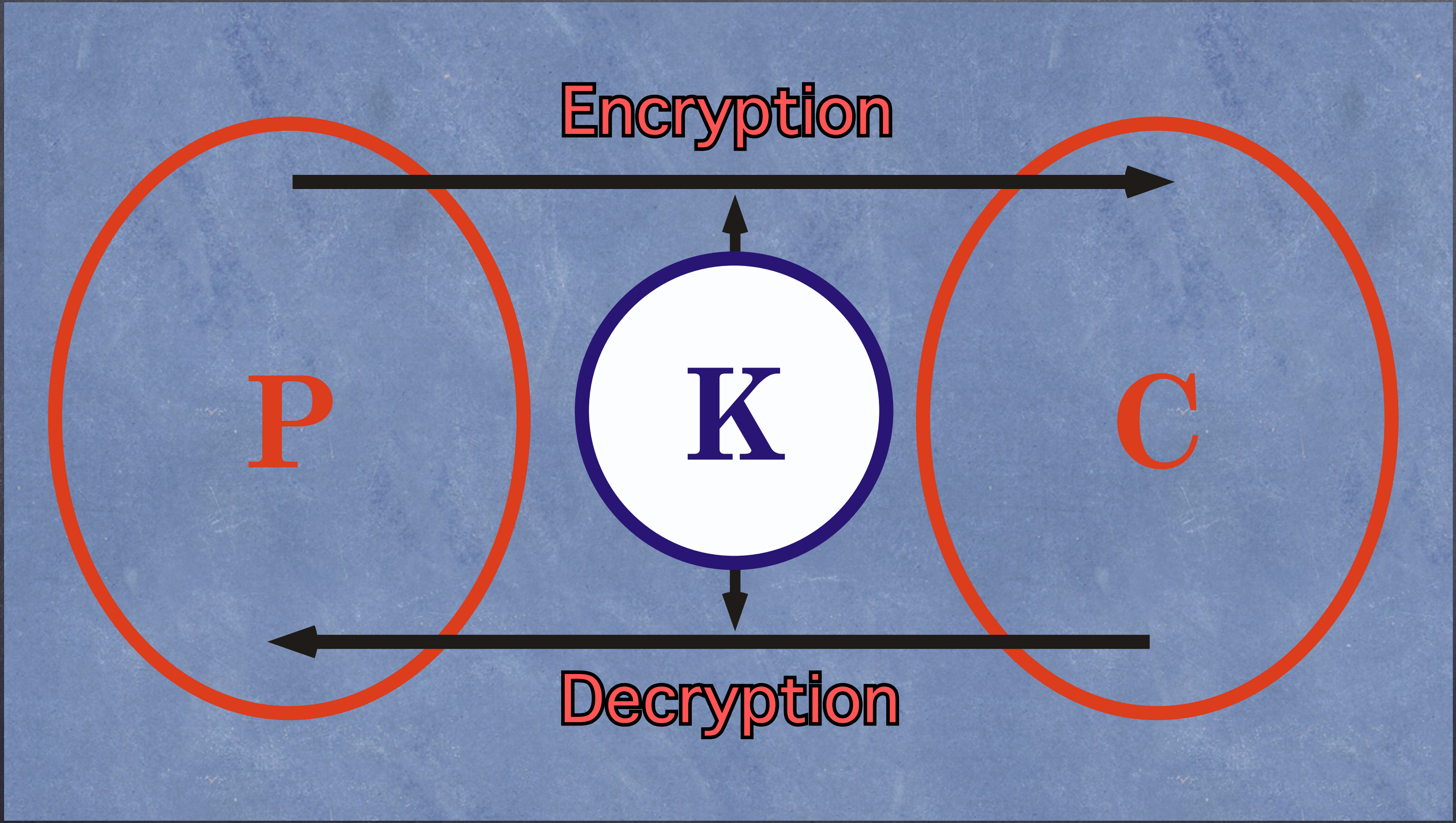
8RdewtU5qkLa$es!T9@

I(D%eXhDqliykl#2cV7dEwnMs

H&fs@tyHvFGhaOKpTrGbl.Z/rUih*

B7B3tdsjUila

Symmetric Encryption



Information Theoretical Security

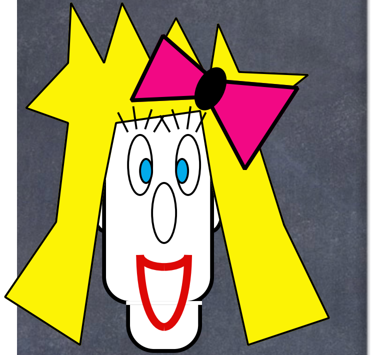
Symmetric Encryption

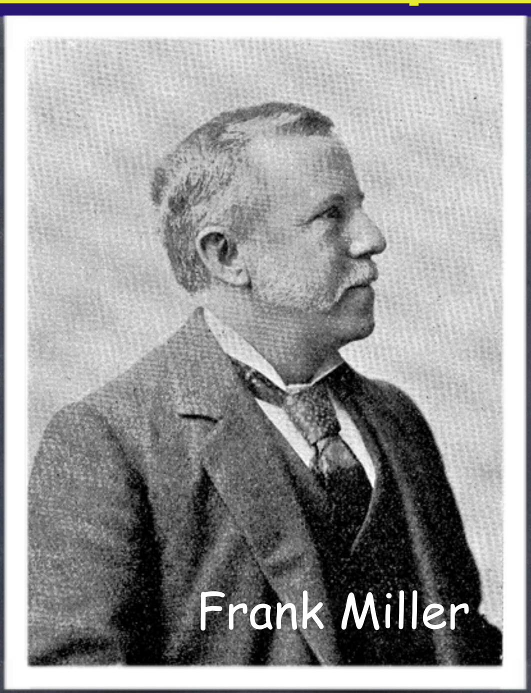
Ceasar's Cipher

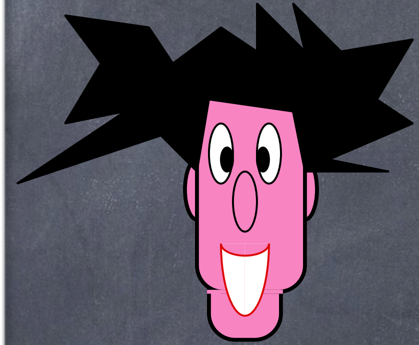
VERNAM's Cipher

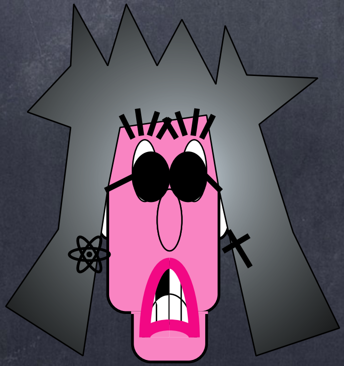
m

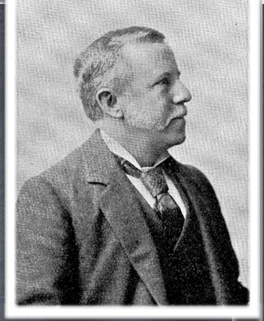
1
0
1
0
0
1
0
0
1
1
1
1
1
0
0
1







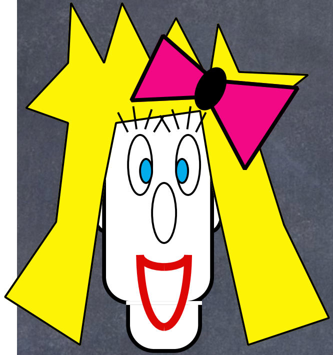


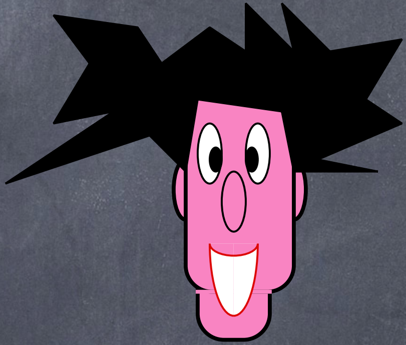


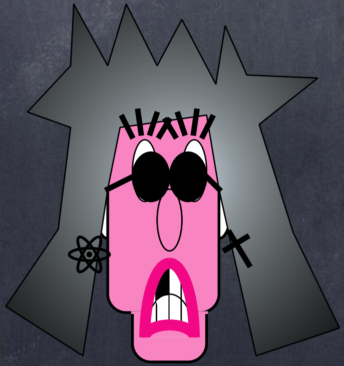
VERNAM's Cipher

m⊕k

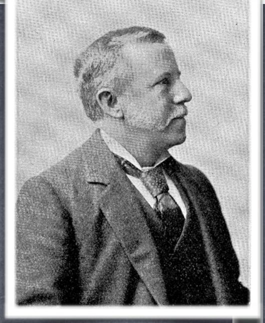
1		1
0		1
1		1
0		0
0		0
1		1
0		1
0		0
1	⊕	1
1		1
1		0
1		1
1		0
0		1
0		1
1		1





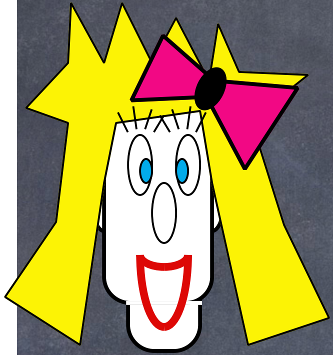


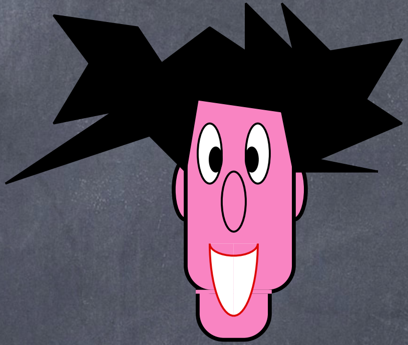
VERNAM's Cipher

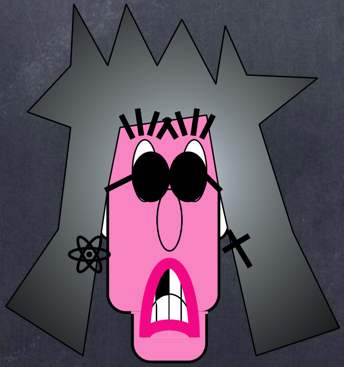


m	$\oplus$k=	c
1	1	0
0	1	1
1	1	0
0	0	0
0	0	0
1	1	0
0	1	1
0	0	0
1	1	0
1	1	0
1	0	1
1	1	0
1	0	1
0	1	1
0	1	1
1	1	0

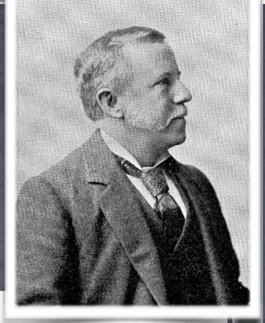
$\oplus =$





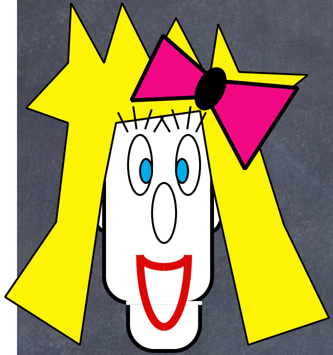


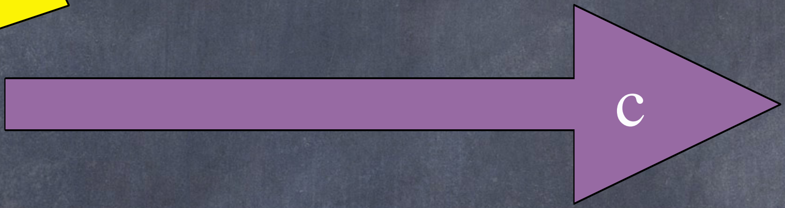
VERNAM's Cipher

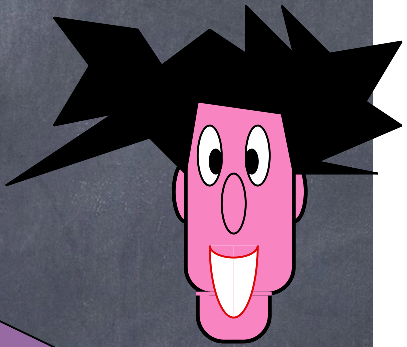


m	⊕k=	c
1	1	0
0	1	1
1	1	0
0	0	0
0	0	0
1	1	0
0	1	1
0	0	0
1	1	0
1	1	0
1	0	1
1	1	0
1	0	1
0	1	1
0	1	1
1	1	0

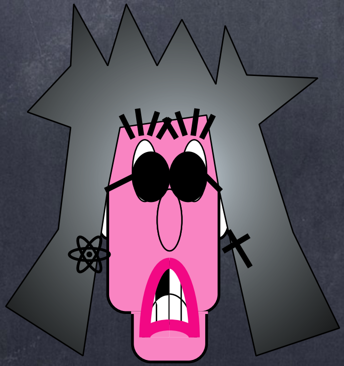
⊕ =



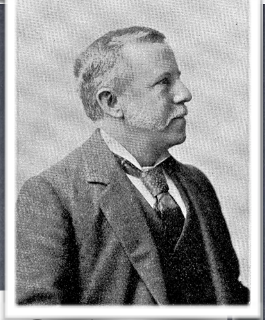




c
0
1
0
0
0
0
1
0
0
0
1
0
1
1
1
0

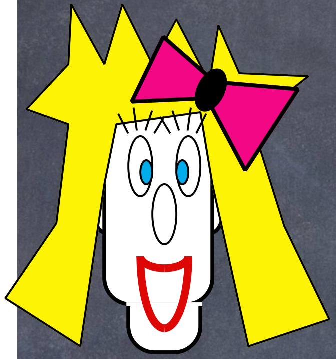


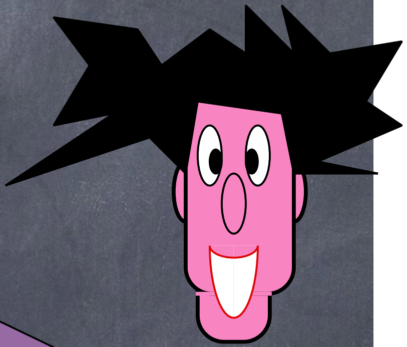
VERNAM's Cipher

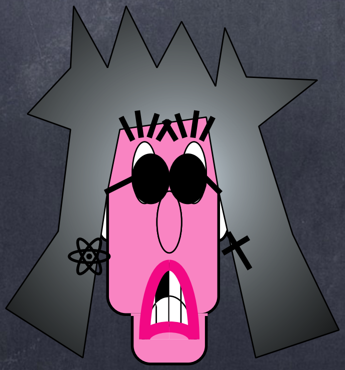


m	⊕k=	c
1	1	0
0	1	1
1	1	0
0	0	0
0	0	0
1	1	0
0	1	1
0	0	0
1	1	0
1	1	0
1	0	1
1	1	0
1	0	1
0	1	1
0	1	1
1	1	0

⊕ =







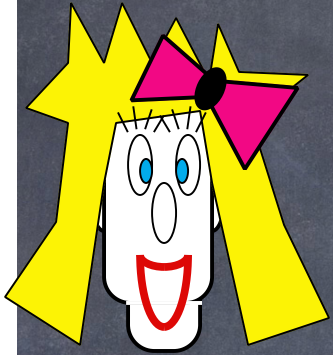
c ⊕	k
0	1
1	1
0	1
0	0
0	0
0	1
1	1
0	0
0	1
0	1
1	0
0	1
1	0
1	1
1	1
0	1

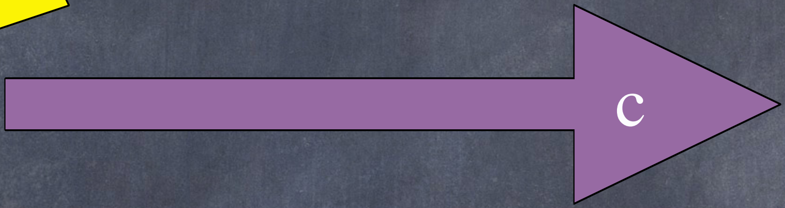
⊕

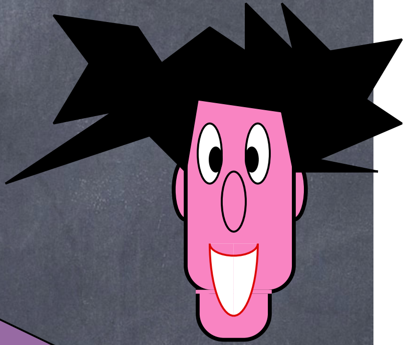
VERNAM's Cipher

m⊕	k=	c
1	1	0
0	1	1
1	1	0
0	0	0
0	0	0
1	1	0
0	1	1
0	0	0
1	1	0
1	1	0
1	0	1
1	1	0
1	0	1
0	1	1
0	1	1
1	1	0

⊕ =

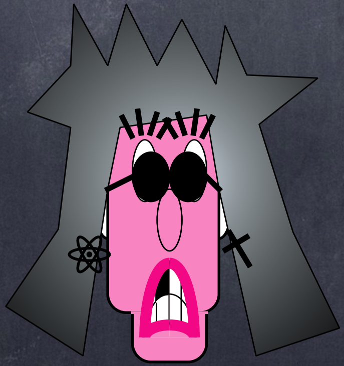






c ⊕	k=	m
0	1	1
1	1	0
0	1	1
0	0	0
0	0	0
0	1	1
1	1	0
0	0	0
0	1	1
0	1	1
1	0	1
0	1	1
1	0	1
1	1	0
1	1	0
0	1	1

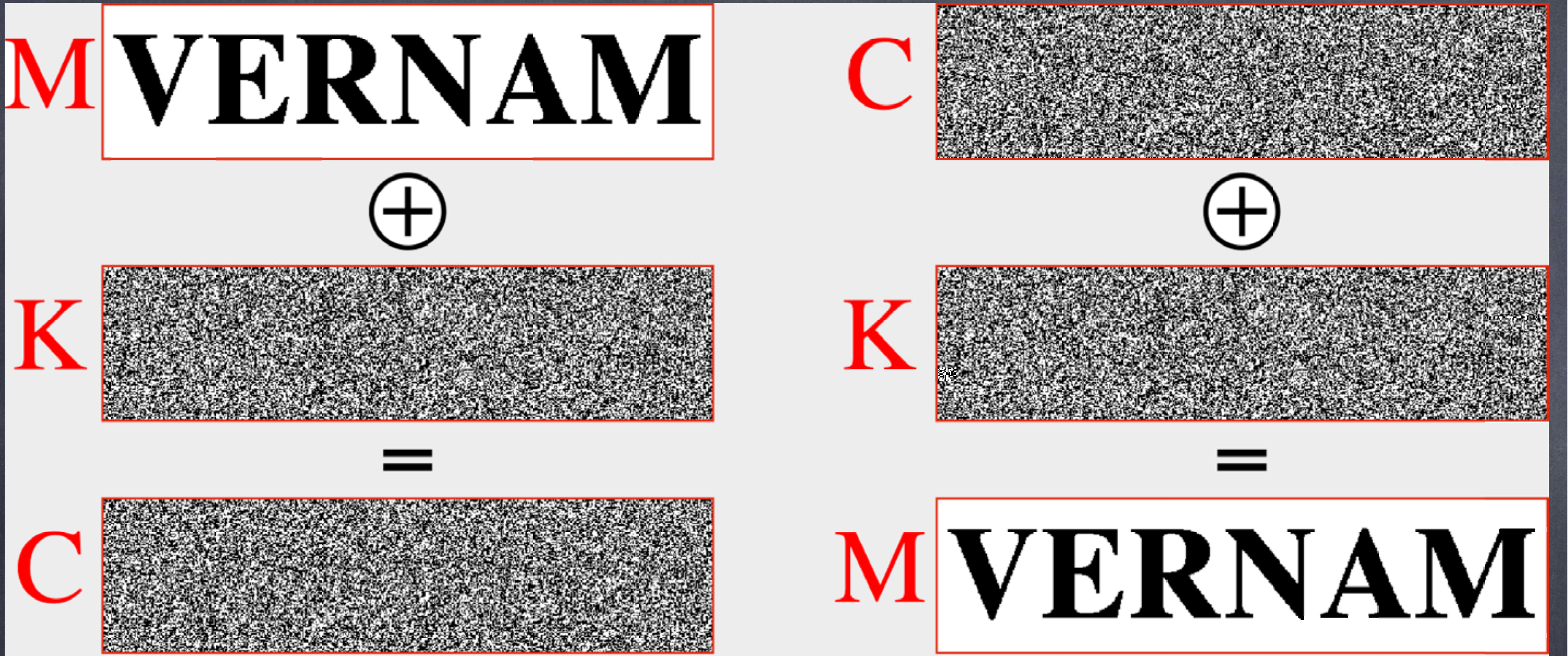
⊕ =



[redacted]

[redacted]

[redacted]



M VERNAM

⊕

K

=

C

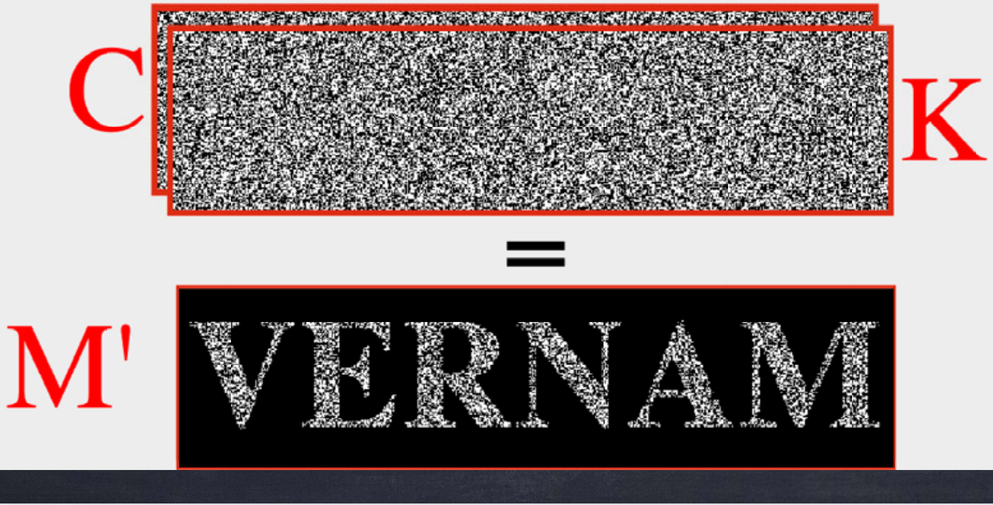
C

⊕

K

=

M VERNAM

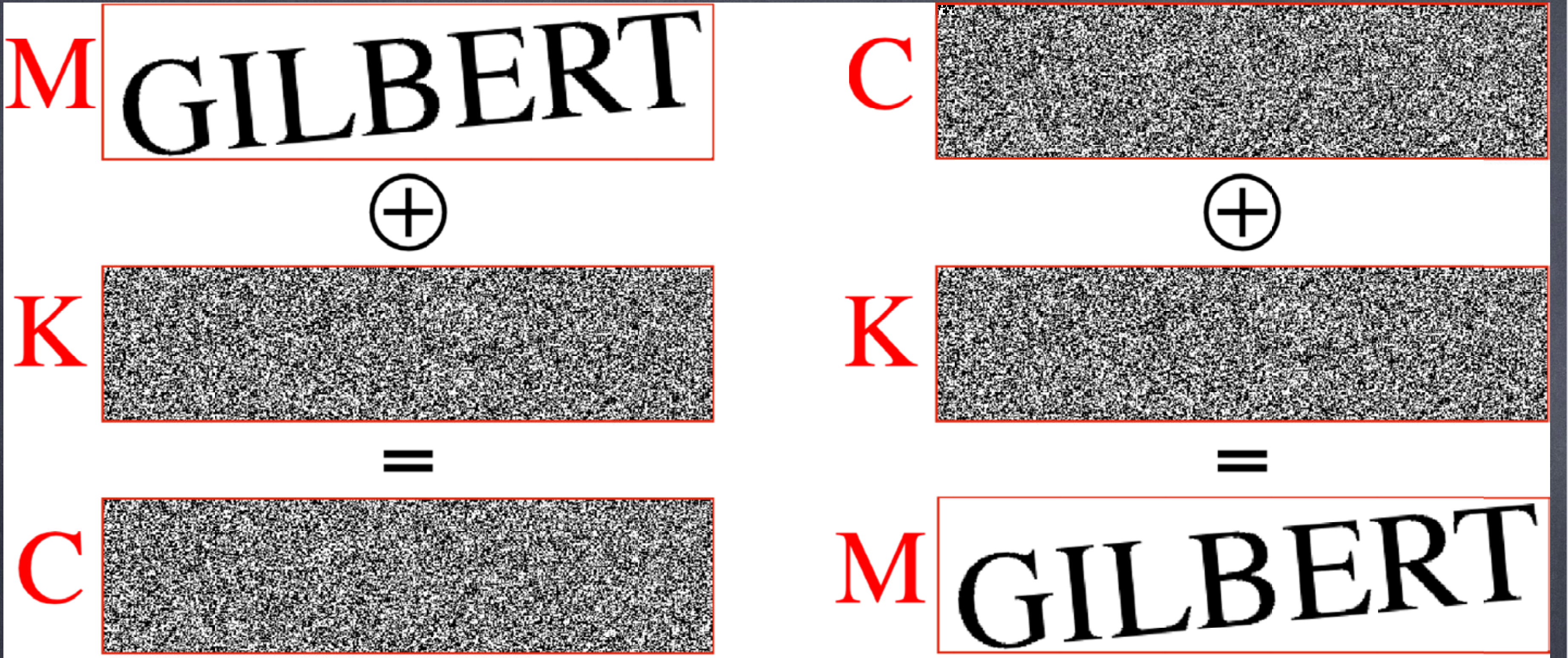


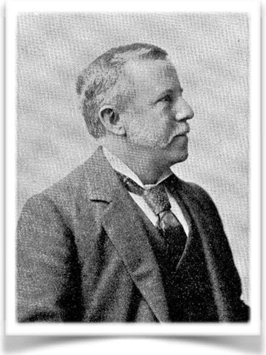
C

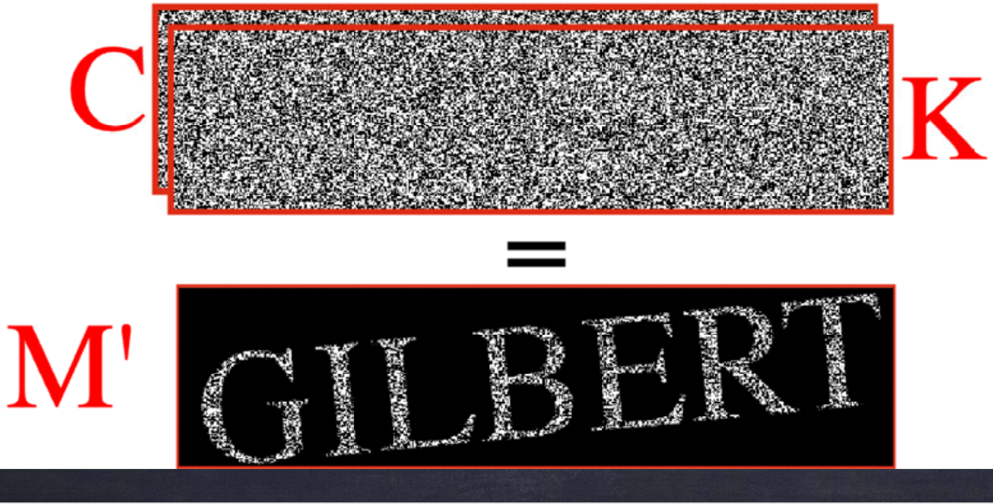
K

=

M' VERNAM


M
GILBERT
⊕
K
=
C
C
⊕
K
=
M
GILBERT

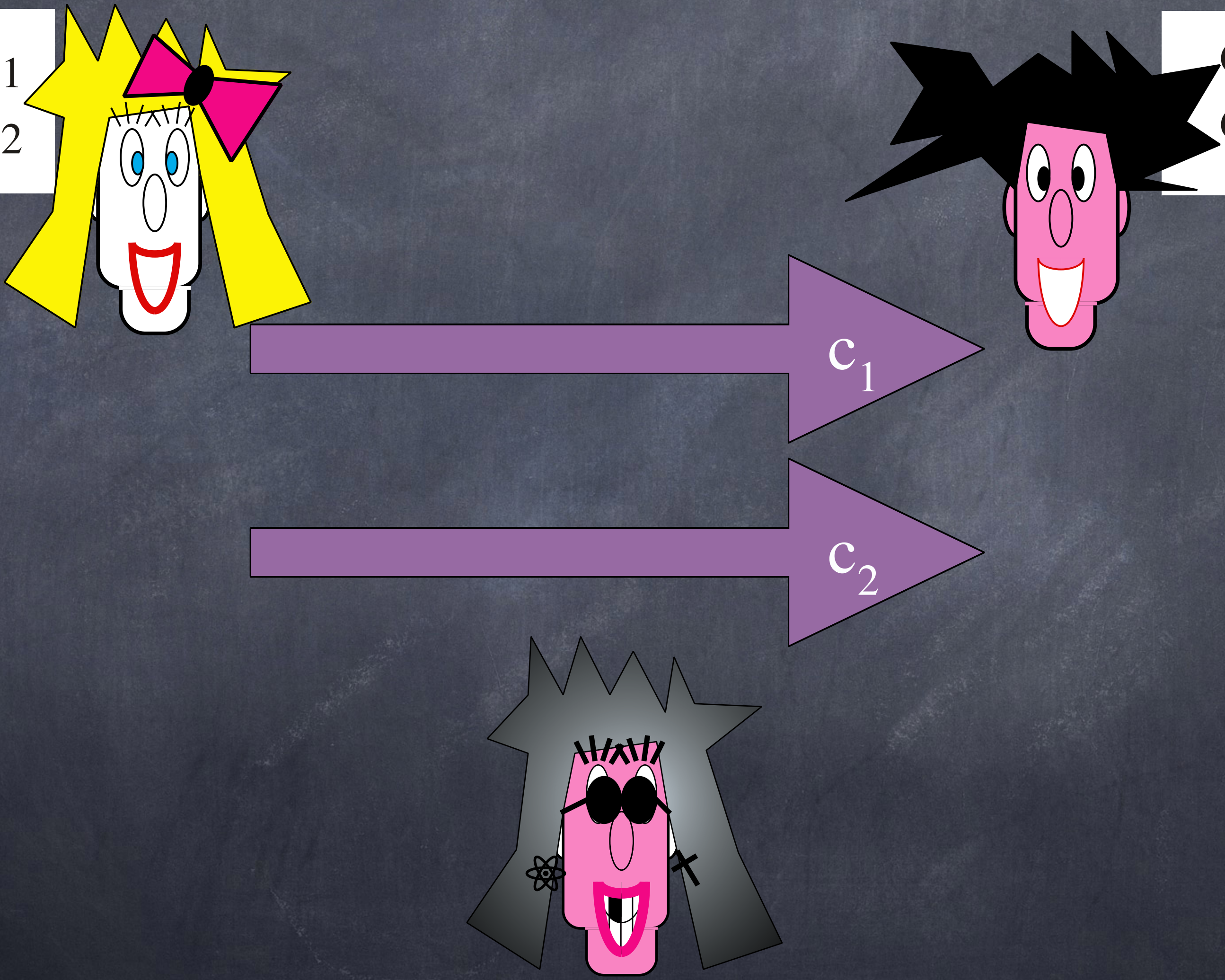



C
K
=
M'
GILBERT

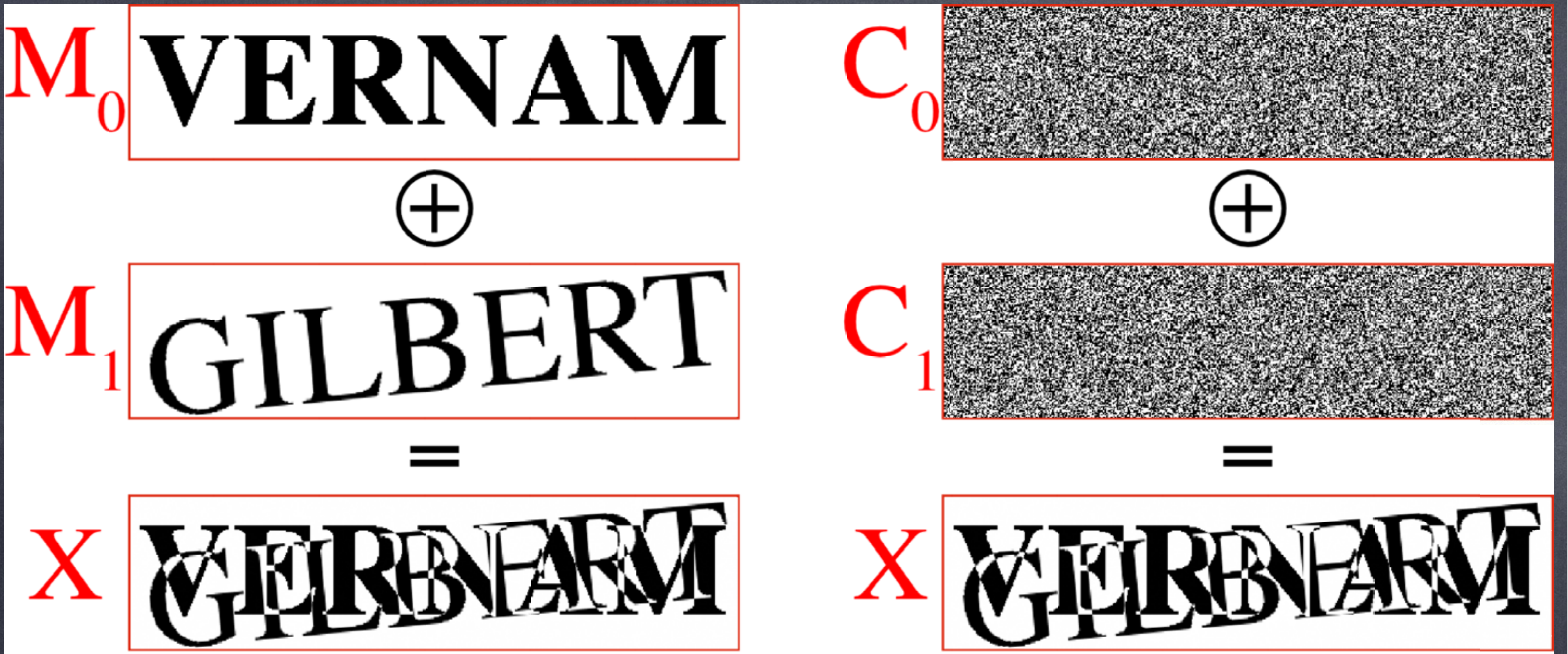
VERNAM's One-Time Pad

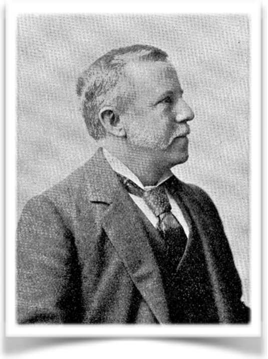
$m_1 \oplus k = c_1$
$m_2 \oplus k = c_2$

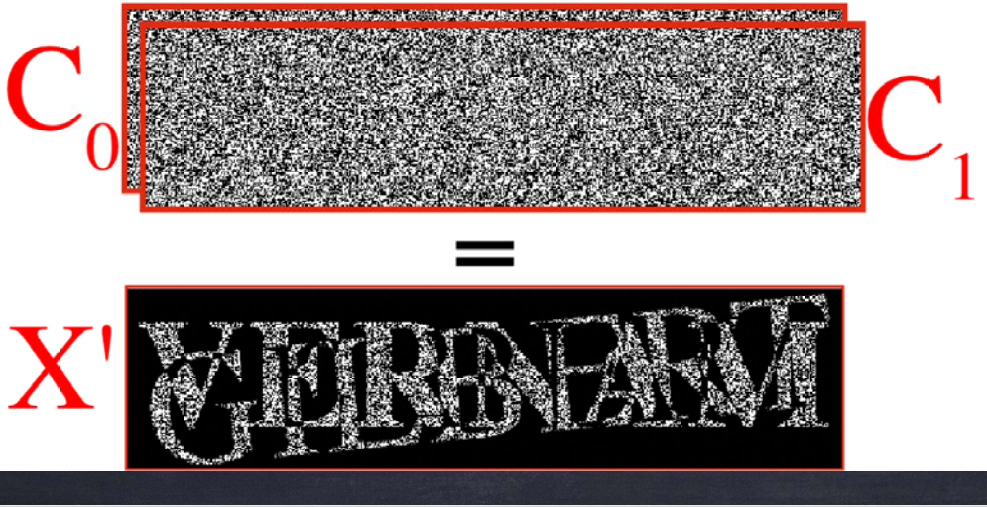
$c_1 \oplus k = m_1$
$c_2 \oplus k = m_2$

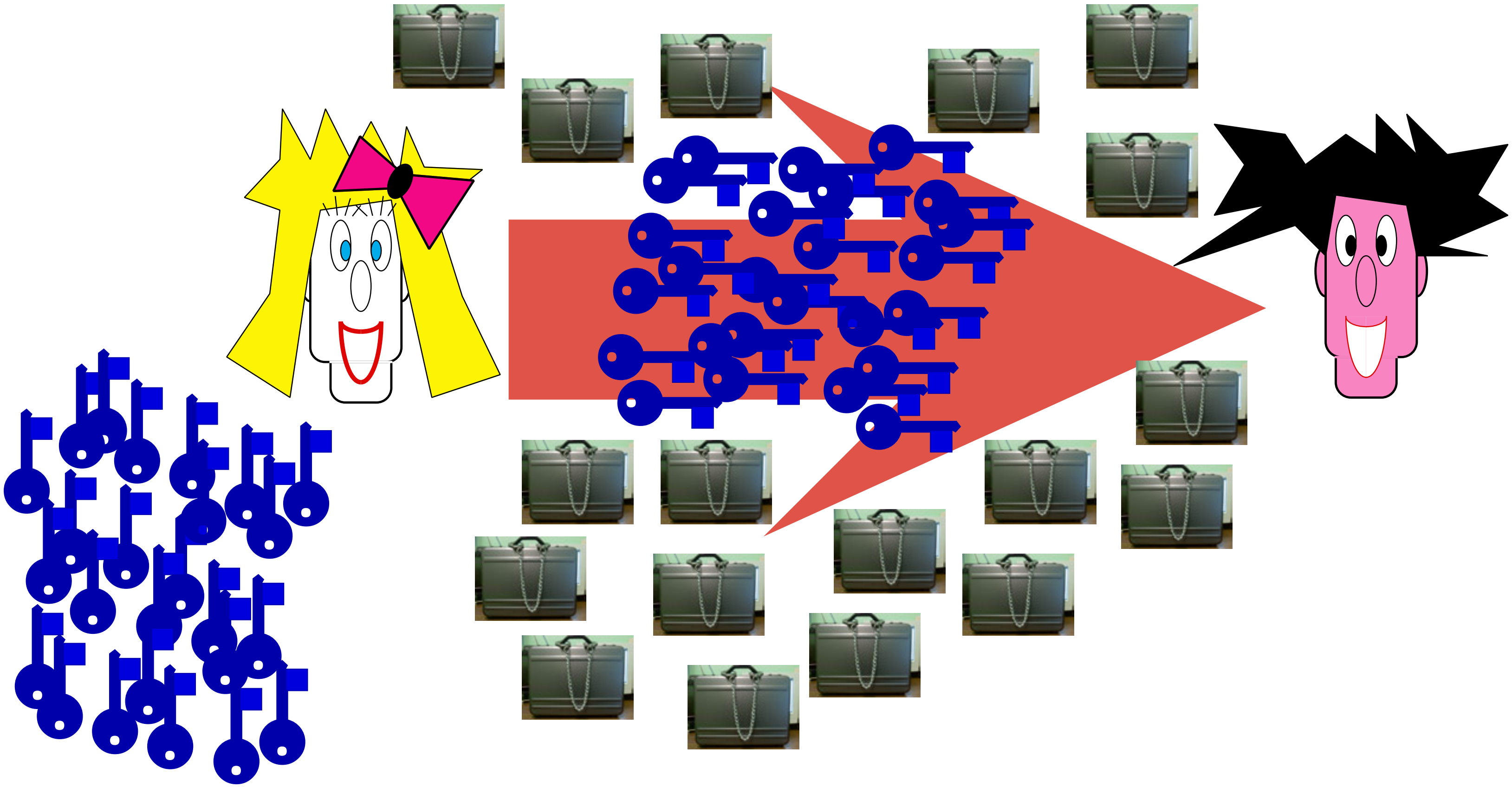


$c_1 \oplus c_2 = m_1 \oplus m_2$

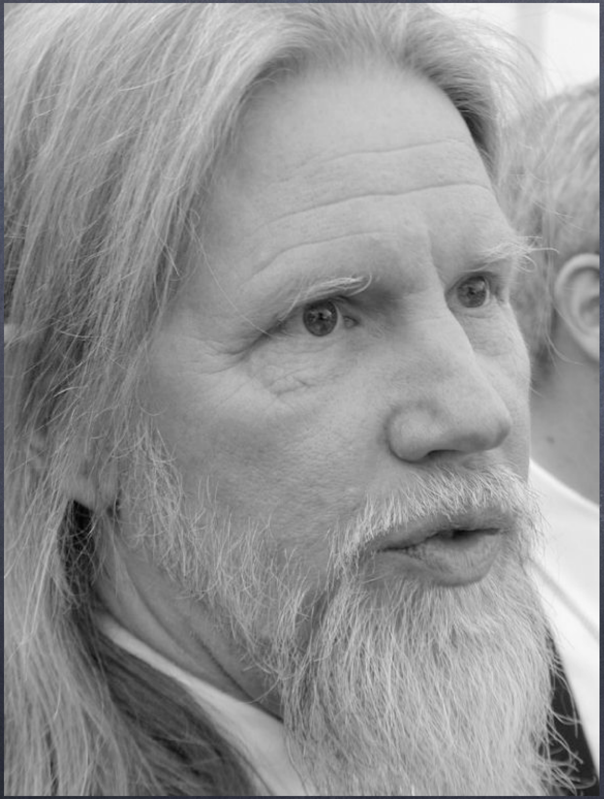

M_0
VERNAM
$\oplus$
M_1
GILBERT
=
X
C_0
$\oplus$
C_1
=
X




C_0
C_1
=
X'

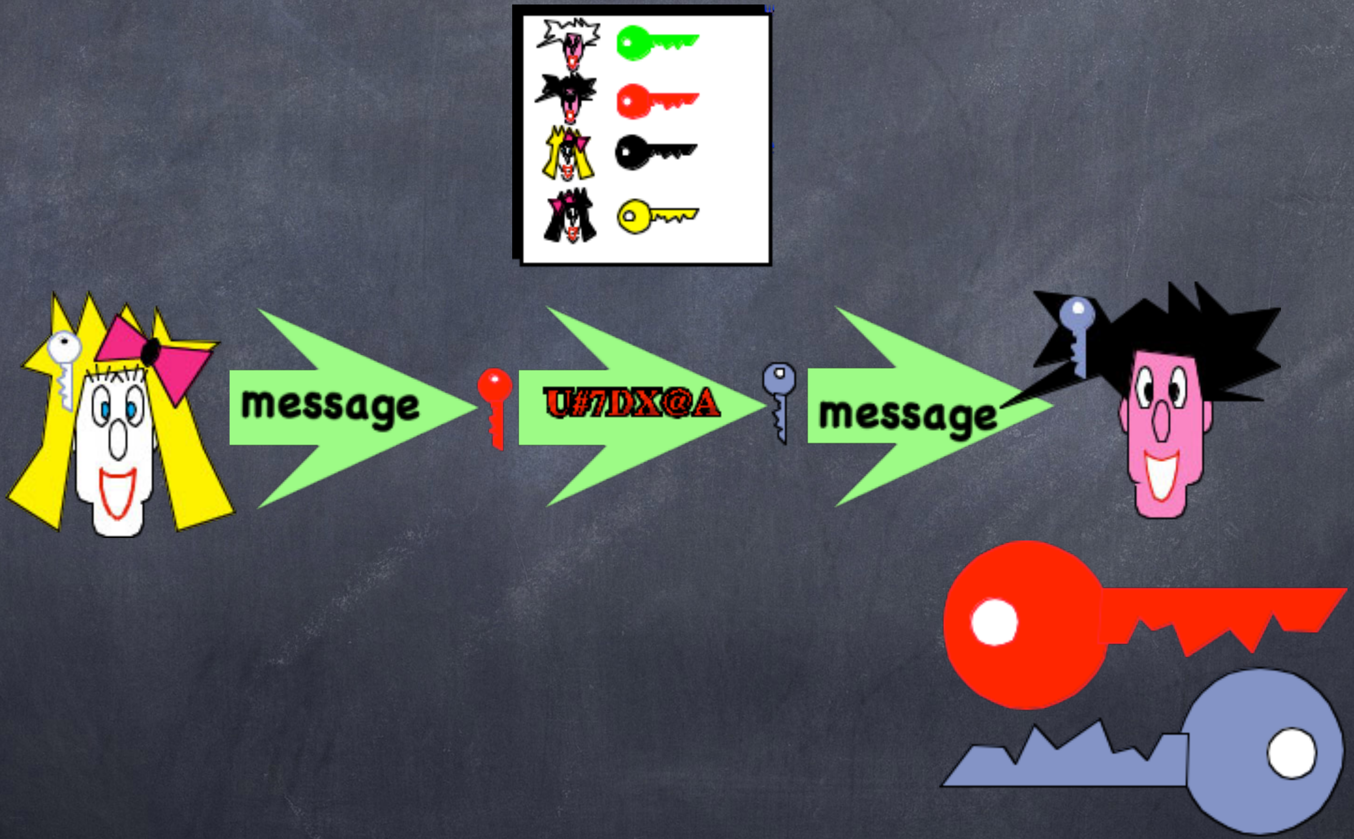


The Public-Key Revolution



Whitfield Diffie and Martin Hellman

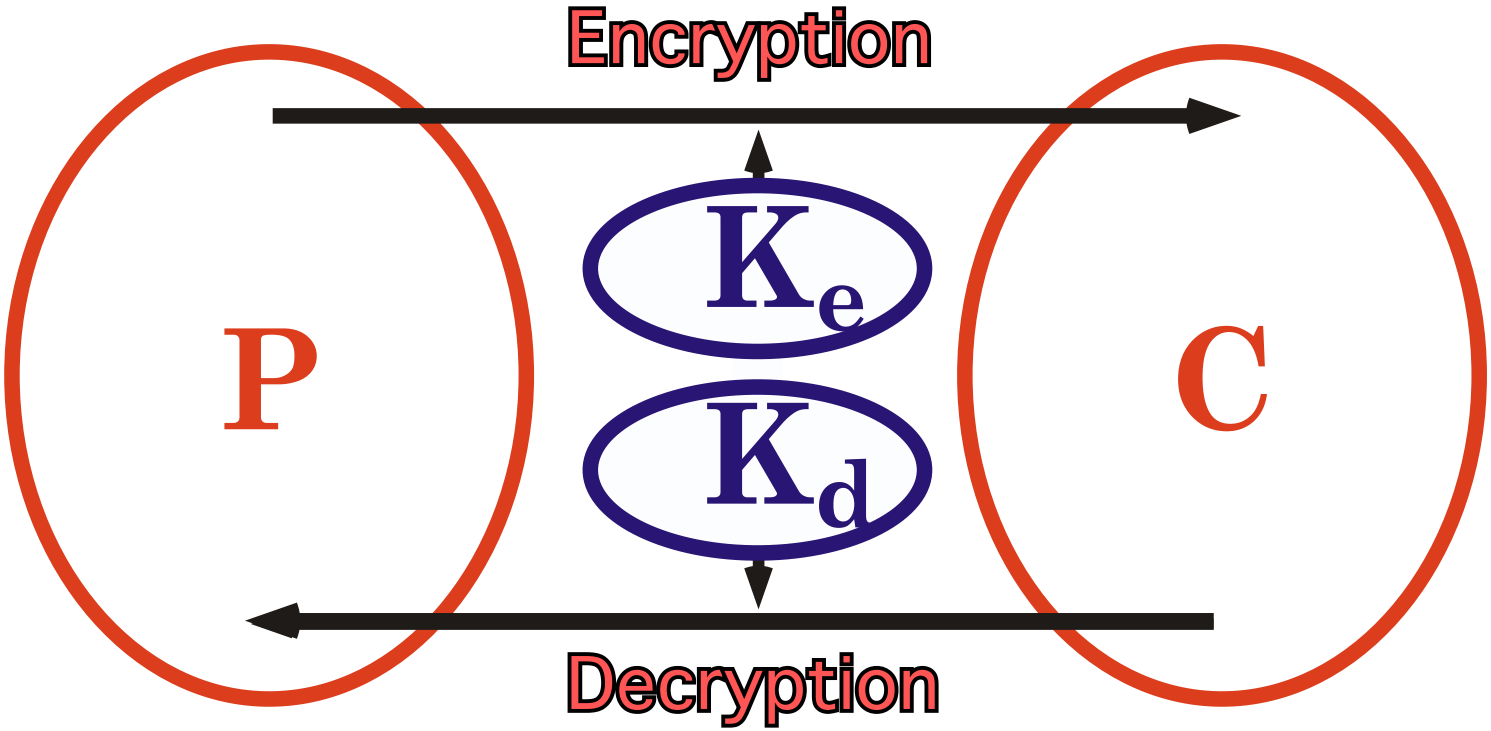
The Public-Key Revolution



Public Key Encryption

Asymmetric Encryption

(Public-Key Cryptography)



Complexity Theoretical Security

RSA Encryption

Public inventors

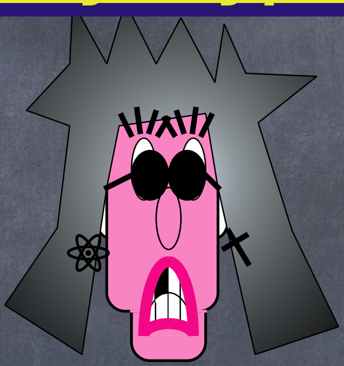
Private inventors

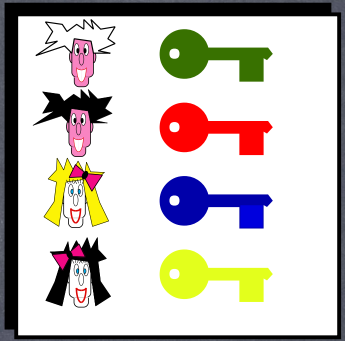
Ellis,

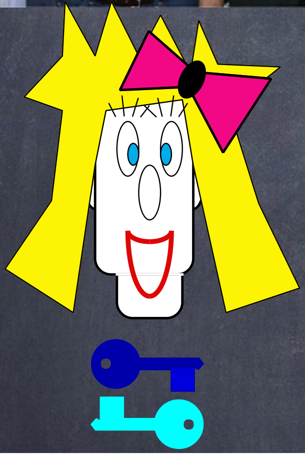
Cocks,

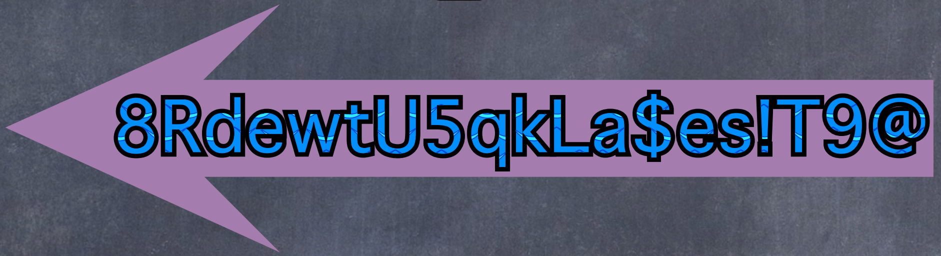
Williamson

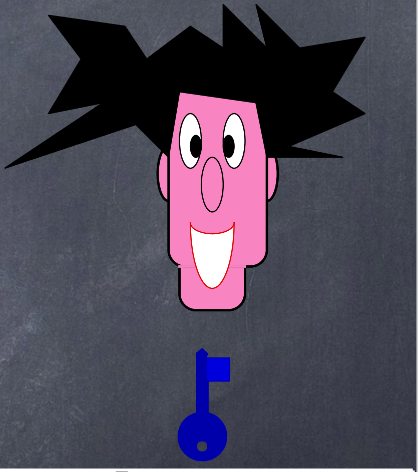
Public-Key Cryptography

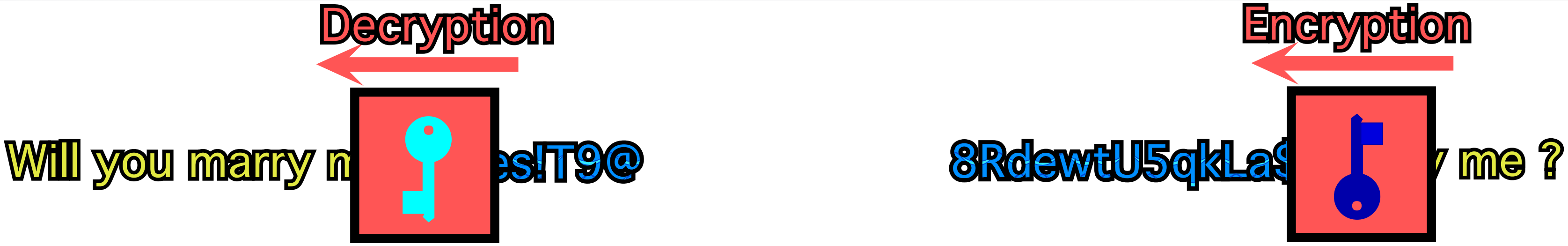












13.99 Primality Testing

Computing mod N

Elementary Operations. Let N,a,b be n-bit integers.

- a+b mod N is computable in time O(n).
- axb mod N is computable in time $O(n^2)$ and asymptotically $O(n^{1+\partial})$.
- a^b mod N is computable in time $O(n^3)$ and asymptotically $O(n^{2+\partial})$.
- gcd(a,b) is computable in time $O(n^2)$.
- [AKS2002]
  Deciding if a number N is prime or not is computable in time $O(n^{12})$.
  Way too slow in practice.
- [PL2005]
  Deciding if a number N is prime or not is computable in time $O(n^6)$.
  Still too slow in practice.

Computing mod N

Rabin-Miller pseudo-primality test. Let N,1<a<N be n-bit integers.

Let $N-1 = 2^s t$ where t is odd.	$O(n)$
Let a be a random element such that $1 < a < N$.	$O(n)$
If $\gcd(a,N) > 1$ **then** fail.	$O(n^2)$
Compute $x_0 := N-1$; $x_1 := a^t \bmod N$.	$O(n^3)$
Compute $x_{i+1} := x_i^2 \bmod N$, for $1 \le i \le s$.	$O(n^2)$
If $x_{s+1} > 1$ **then** fail.	$O(1)$
Let m be such that $x_m > 1$ and $x_{m+1} = 1$.	$O(n)$
If $x_m = N-1$ **then** succeed **else** fail.	$O(1)$

- Rabin theorem[1977]. Let N,a be n-bit integers.

- **If** N is prime **then** all a such that gcd(a,N)=1 lead to success
- **else** at least 3/4 of all a such that gcd(a,N)=1 lead to failure.

Computing mod N

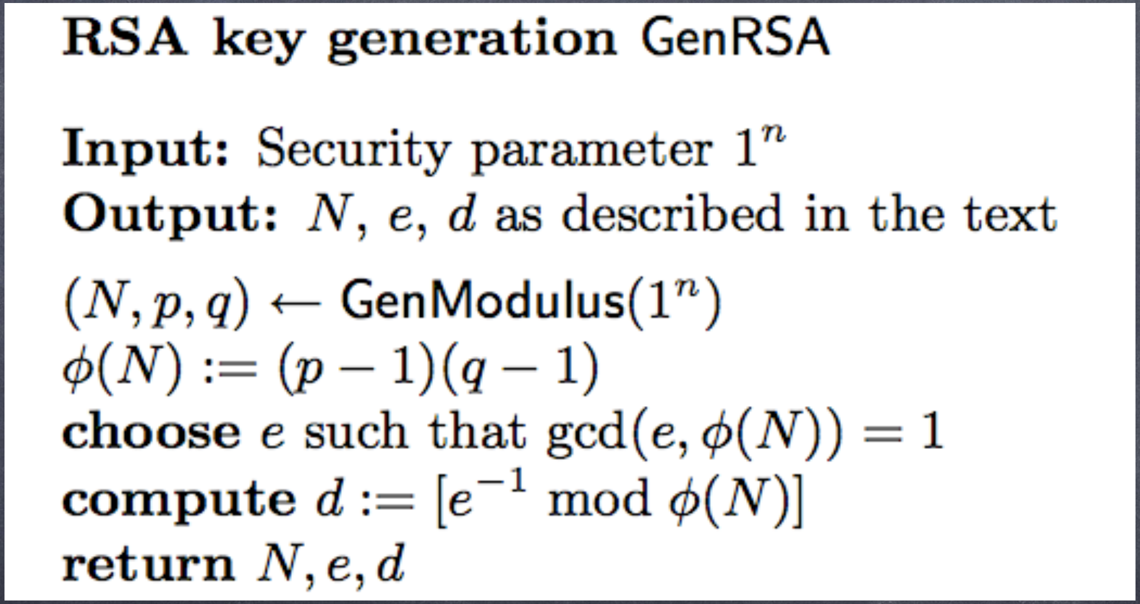
Rabin theorem[1977]. Let N,a be n-bit integers.

- **If** N is prime **then** all a such that gcd(a,N)=1 lead to success
- **else** at least 3/4 of all a such that gcd(a,N)=1 lead to failure.

Corollary. If this test is executed k times with random independent a's, then if N is prime then Pr[k success] = 1 else Pr[k success] $< 1/4^k$.

Running time = $O(kn^{2+\partial})$

RSA Encryption



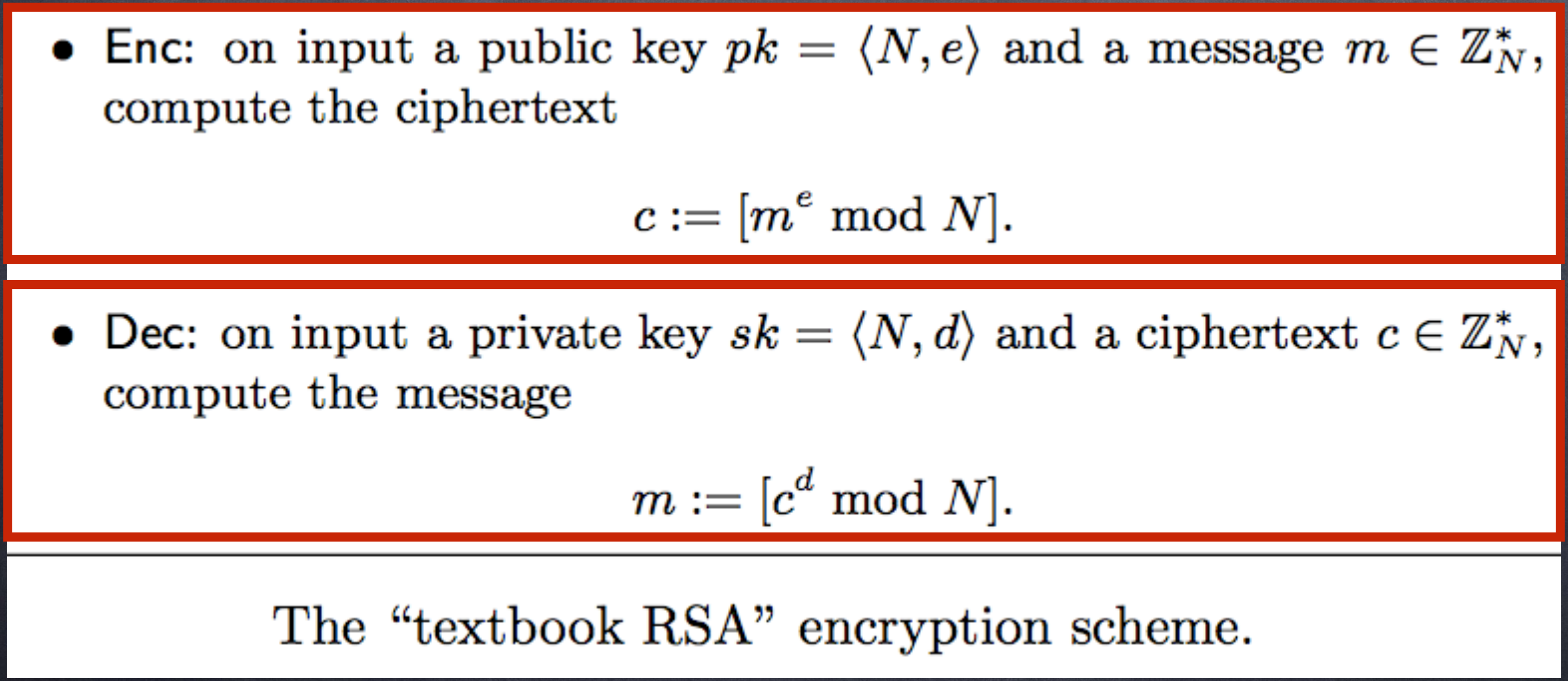
RSA key generation GenRSA

Input: Security parameter 1^n
Output: N, e, d as described in the text

```
$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
choose $e$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return $N, e, d$
```

In Cocks' variation, e=*N* and therefore *d*=*N*$^{-1}$ mod φ(*N*).

RSA Encryption



- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the ciphertext

$$c := [m^e \bmod N].$$

- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message

$$m := [c^d \bmod N].$$

The "textbook RSA" encryption scheme.

The RSA Assumption

The **RSA** problem can be described informally as:

- a modulus N,
- an exponent $\mathbf{e > 0}$ that is relatively prime to $\boldsymbol{\varphi(N)}$, and
- an element $c \in \mathbb{Z}_N^*$,
- compute $\sqrt[e]{c} \bmod N$;

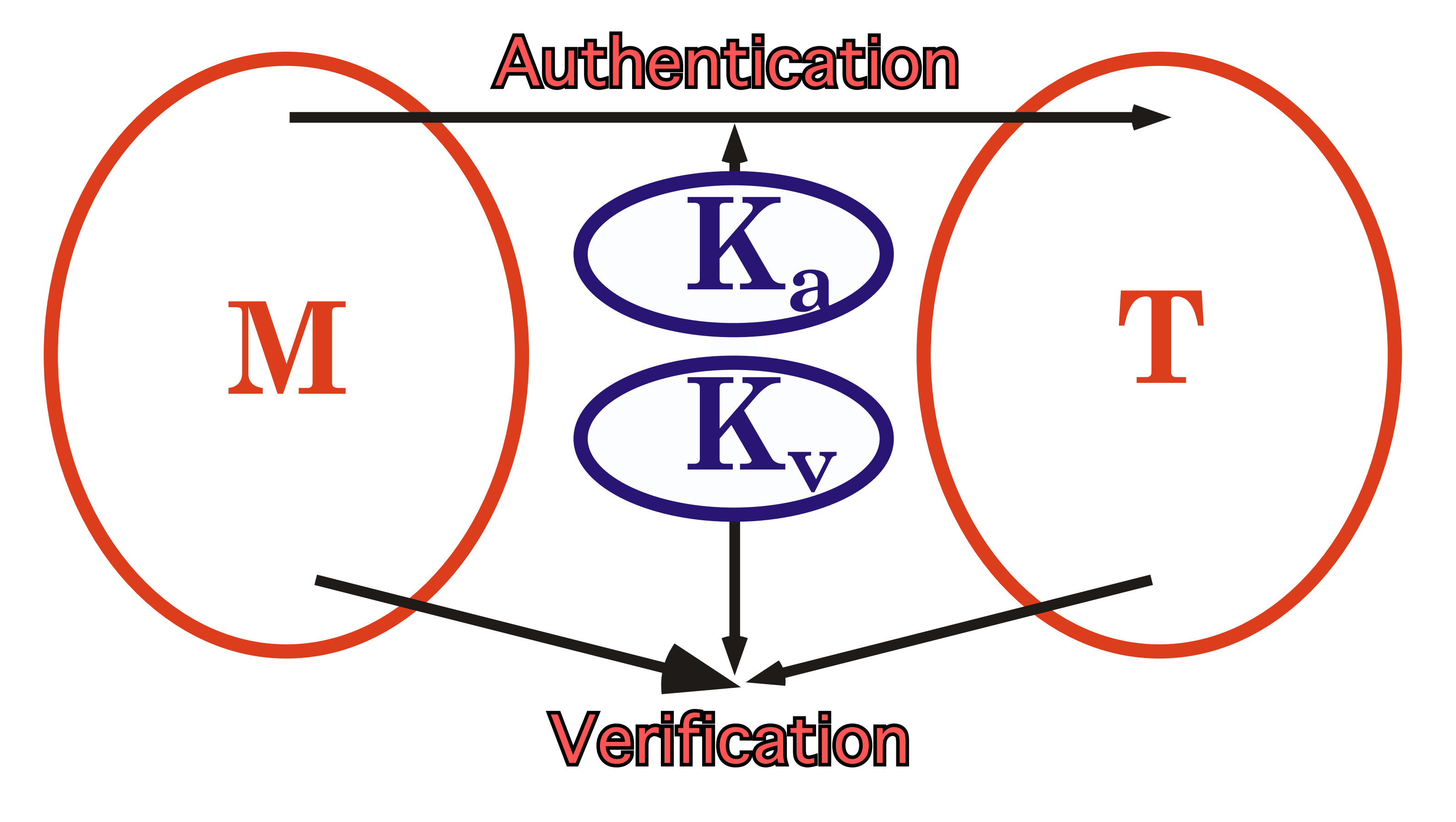
or

Given N,e,c find m such that $m^e = c \bmod N$.

Digital Signatures

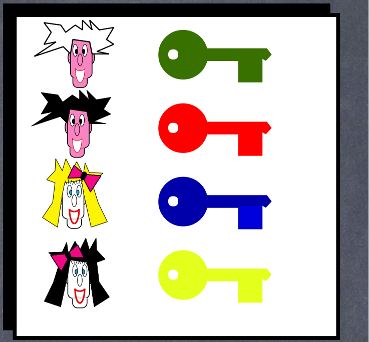
Asymmetric Authentication

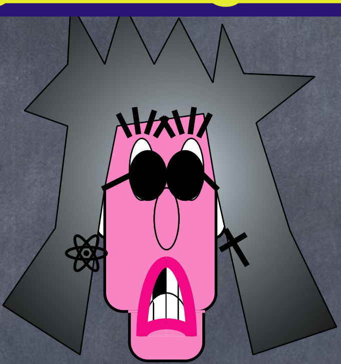
(Digital Signature Scheme)

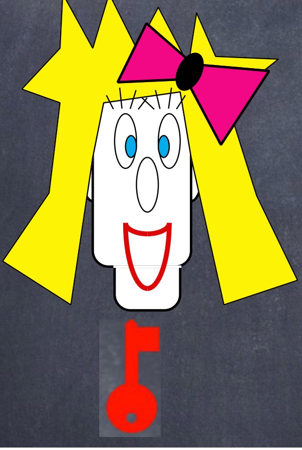


Complexity Theoretical Security

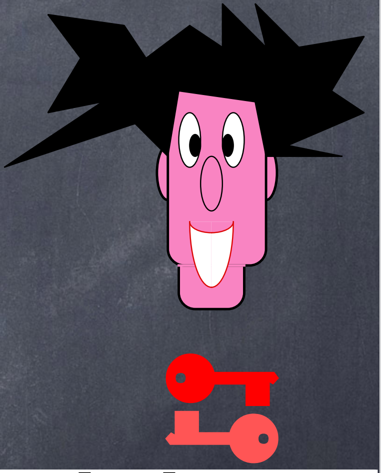
Digital Signature







8RdewtU5qkLa$es!T9@
Will you marry me ?



Verification

VALID es!T9@
ry me ?

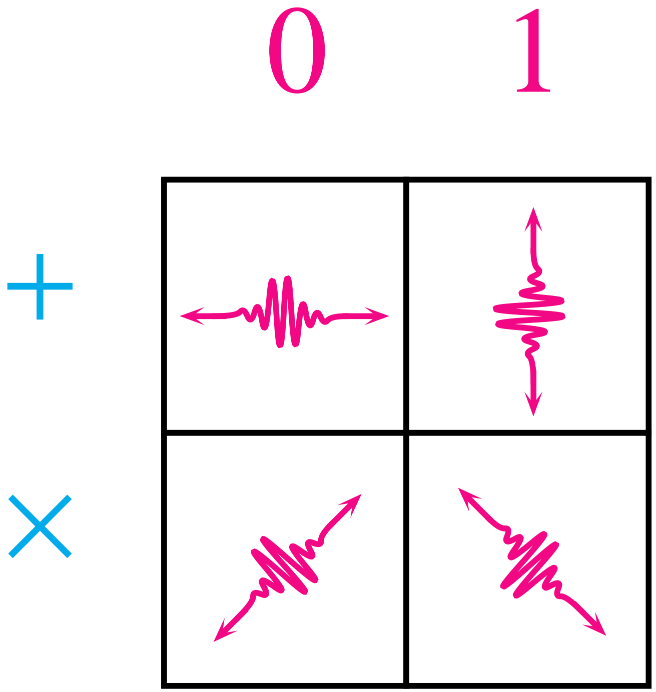
Authentication

8RdewtU5qkLa$ y me ?

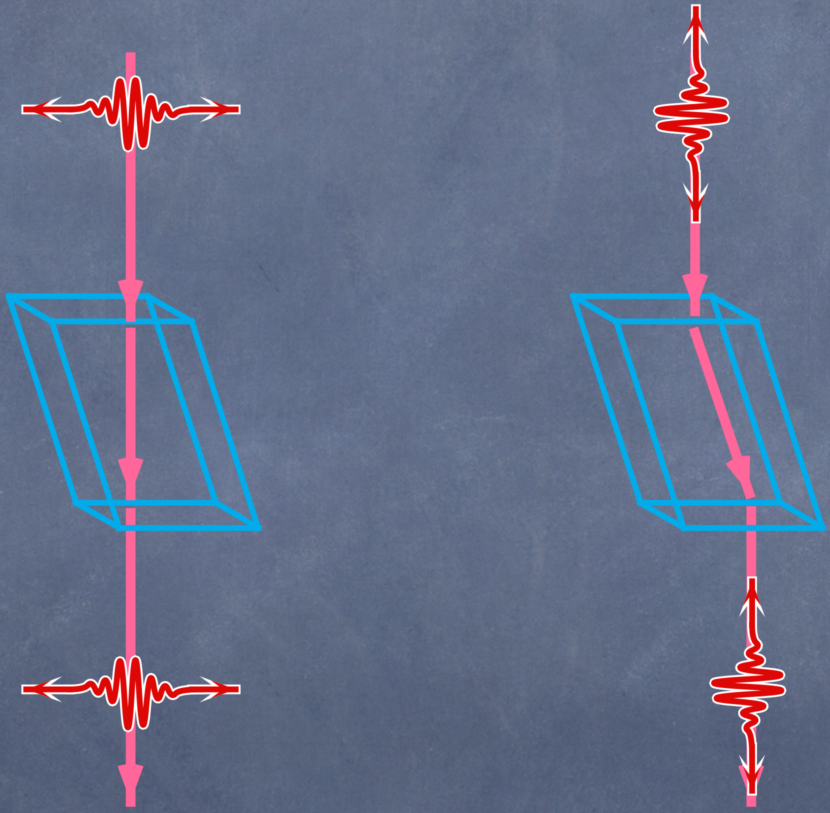
Quantum

Cryptography

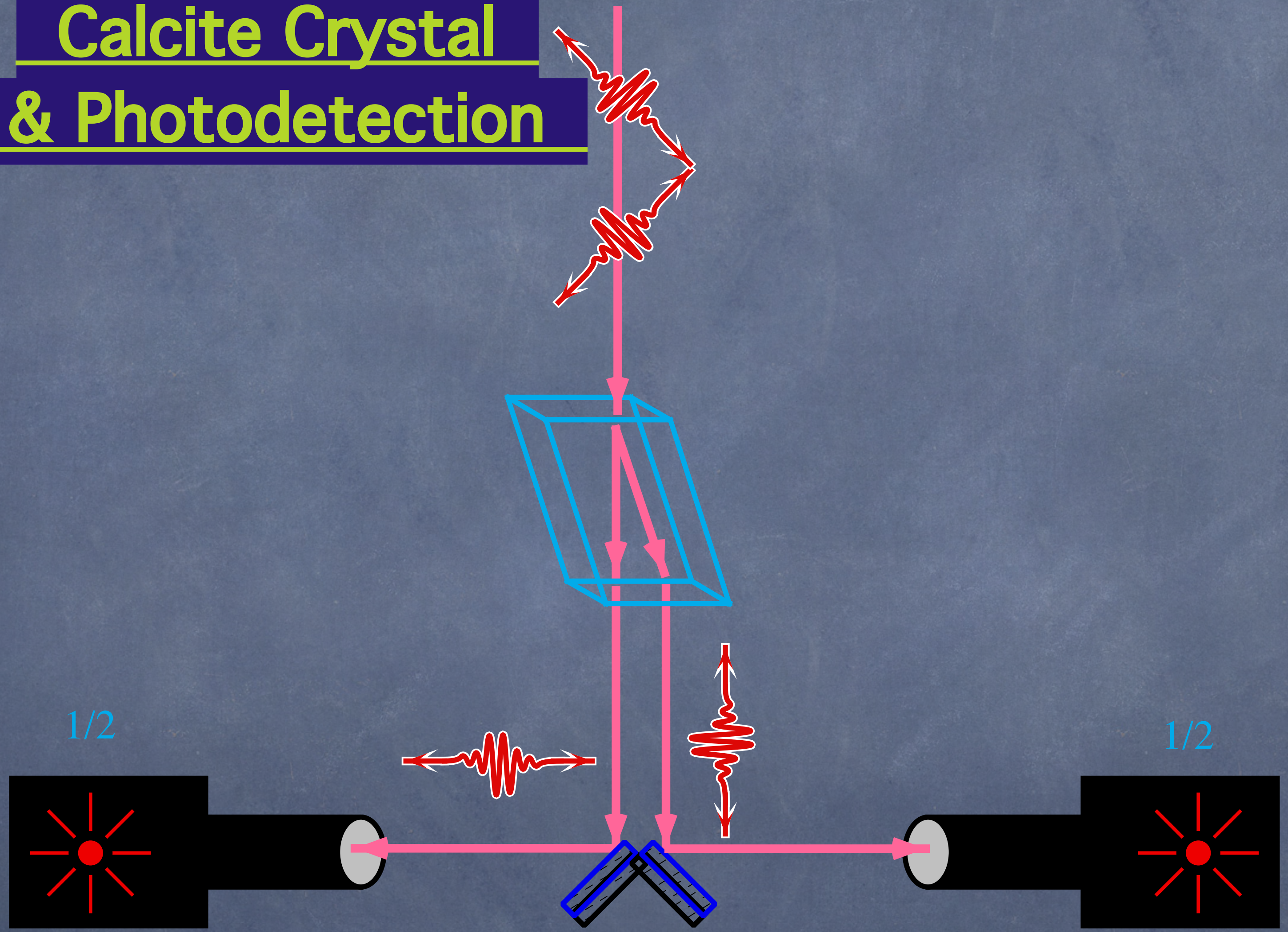
Ambiguous Coding Scheme



Calcite Crystal

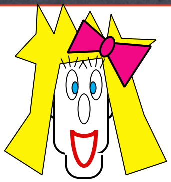


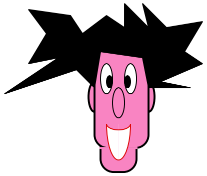
Calcite Crystal & Photodetection



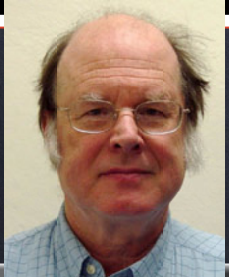
Quantum Key Distribution

Quantum Key Distribution





A:	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	
	×	+	×	+	+	+	×	×	×	×	+	+	+	+	×	×	×	+	×	+	+	+	×	+	
B:	×	×	+	+	×	+	+	+	×	+	+	×	×	×	+	×	×	×	+	+	×	+	×	+	
	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	
A:	×	+	×	+	+	+	×	×	×	×	+	+	+	+	×	×	×	+	×	+	+	+	×	+	
B:	0	👎	👎	0	👎	1	👎	👎	1	👎	0	👎	👎	👎	👎	1	0	👎	👎	1	👎	0	0	0	
B:	0			0		1			1		0					1	0			1		0	0	0	
A:	0			0		1			1		0					1	1			1		0	0	0	
A:	0					1					0						1							0	
B:	=					=					=						≠							=	20%
B:				0					1							1				1		0	0		
A:				0					1							1				1		0	0		



Bennett-Brassard

Quantum Key Distribution

- Produces raw classical key
- Observed error rate indicates amount of eavesdropper information
- Error-correction is used to fix errors
- Random hash function is used to distill a smaller very secret classical key

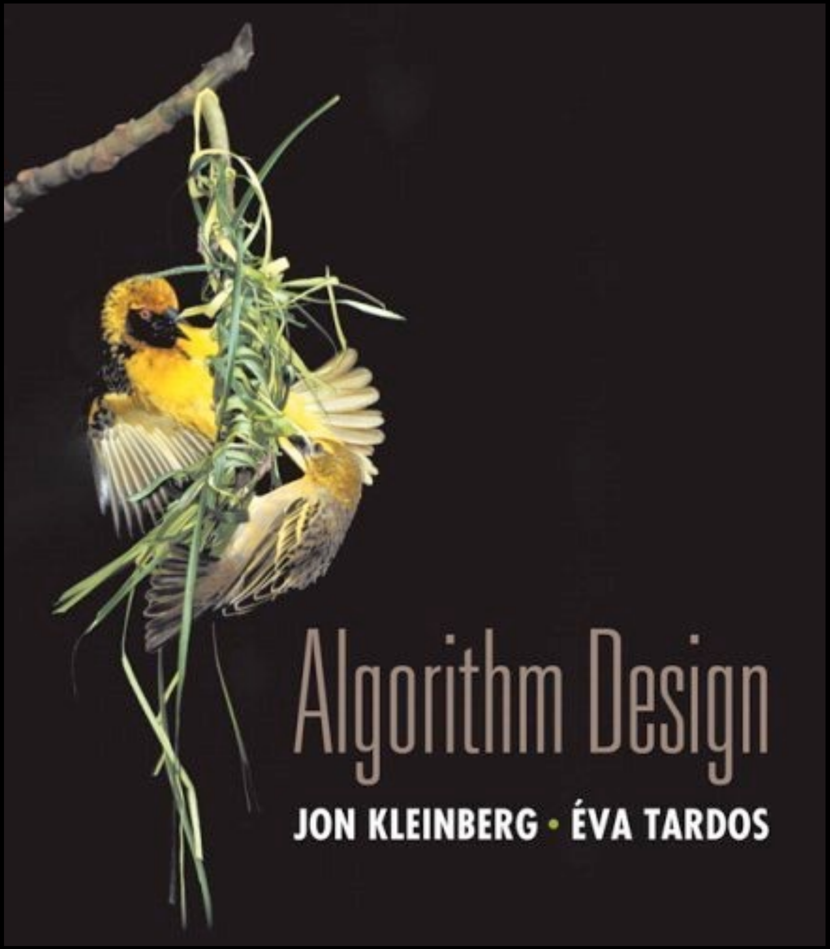
COMP-547B

Cryptography and Data Security

Lecture 01

Prof. Claude Crépeau

School of Computer Science
McGill University



Chapter 13

Randomized Algorithms

PEARSON
Addison Wesley