Data Security:

The art of providing secure communication over insecure channels.

Not a problem that suddenly arose when computers were invented - it’s as old as mankind.

Data Security falls naturally into two cathegories:

* Confidentiality

* Authenticity

CONFIDENTIALITY

A sends a message M to B.

No third party should be able to compute M, ideally not even any partial information on M.

Applications

Databases
Industrial confidential information
Email

Confidentiality is the classical military concern in data security. Almost the only concern up until the 1970-ties.

INTEGRITY

A sends a message M to B.

No third party can change M without this being noticed by B.

- but note that nothing prevents an adversary from just copying a message and repeat it again later. Thus in most cases we want more:

(STRONG) AUTHENTICITY

A sends a message M to B

B gets, in addition to M, also some information PROVING that A sent exactly the message that was received.

If this proof can be tested, not only by B, but by anyone else, we speak of strong authenticity or *non-repudiation* or *digital Signatures*.

Applications:

Electronic Payments
Contracts
Authenticity of info on the net.

How to Implement all this?

We want real solutions, providing Confidentiality and Authenticity.

We start from

Primitives

based these, we build

Crypto Systems and
Authentication systems

based on which we build

Protocols for secure communication

Primitives

-basic objects that can be used to build more complex systems.

Ex: One-way functions:

f: A→► B

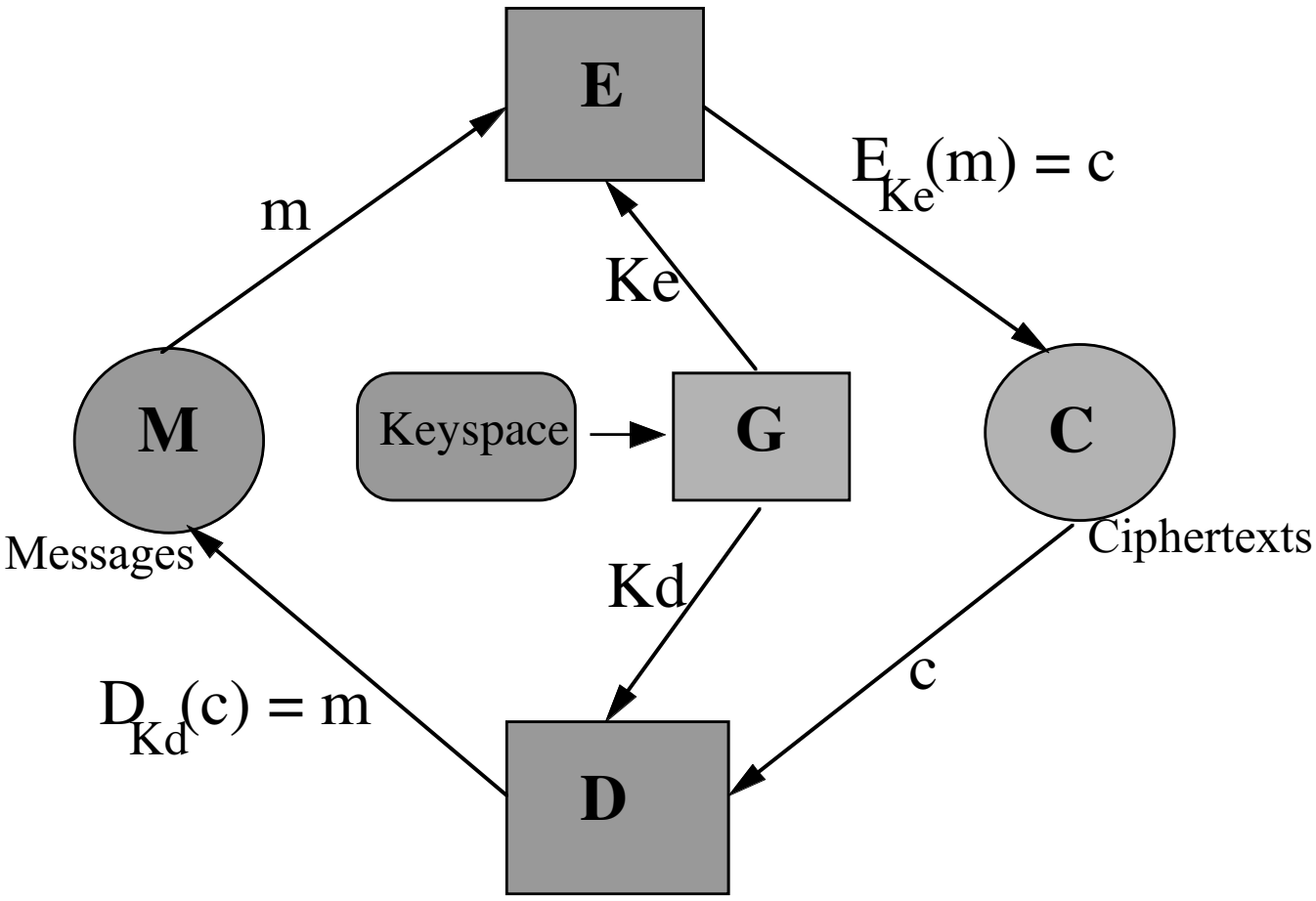
is a one-way function if

from x, f(x) is *easy* to compute, but

given any y in the image of f, it is *hard* to come up with any x, such that y = f(x).

A real definition requires that we define what “easy” and “hard” should mean.

CRYPTOSYSTEM



The system is given by the algorithms E and D for en- and decryption, and the algorithm G for generating keys taken from the keyspace

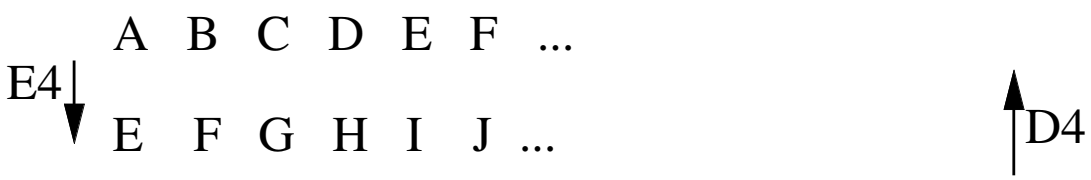
Gievn a key, E and D induce a mapping from M to C, resp. from C to M.
For any pair (Ke,Kd) output by G, it must hold that
m = D(Kd, E(Ke,m))

Conventionelt (Classical System): Ke = Kd.

Public Key System: infeasible to compute Kd from Ke.
==> Ke can be *public* simultaneously with Kd being kept *secret*. Hence the name.

Eks. 1: Cæsar substitution

Key Space: 0,1,2,4...,26
Let us put Ke=Kd=4.



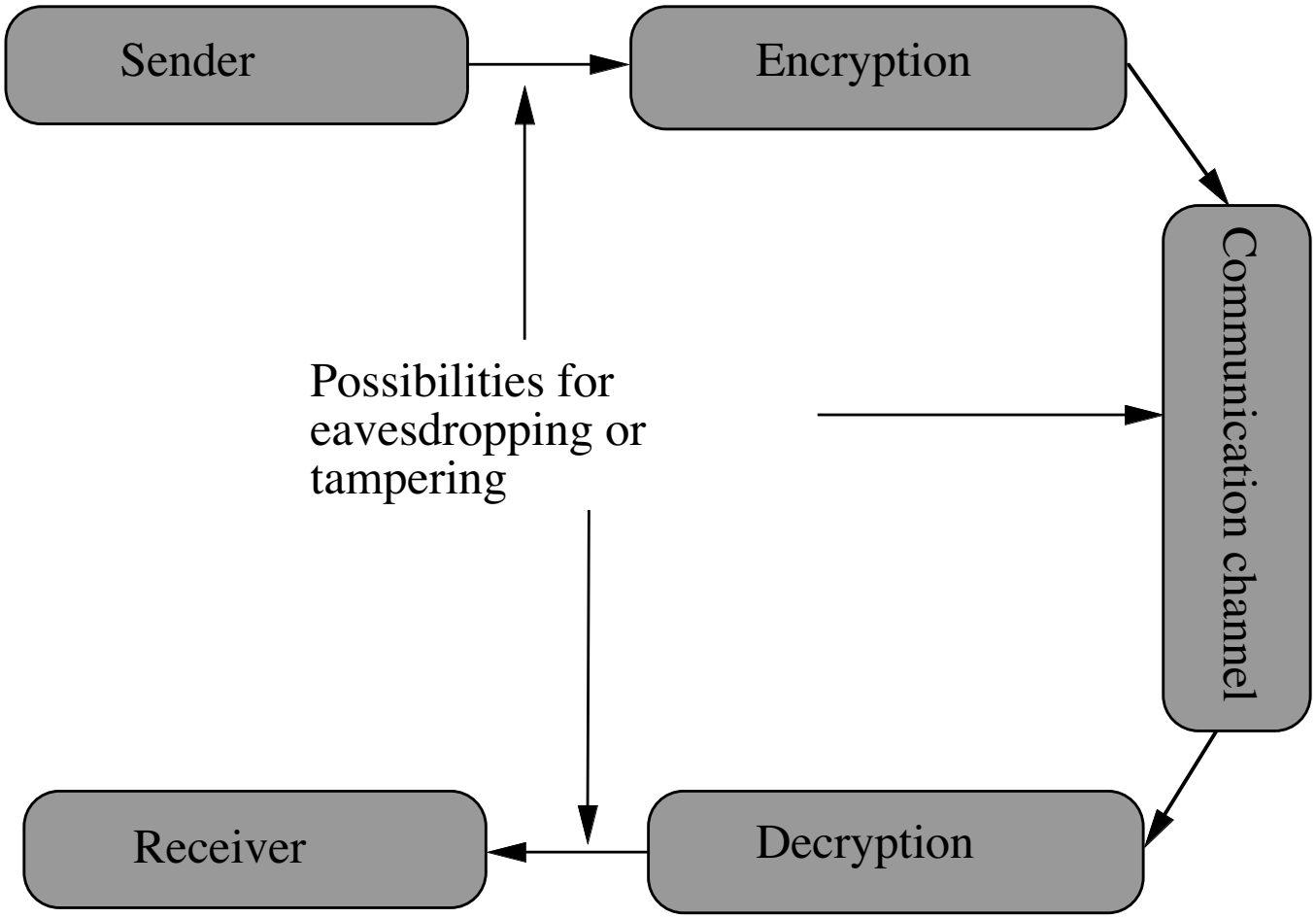
Eks. 2 One Time Pad

Messages and keys are bit strings. The key is chosen at random.

M	0 1 1 1 0 0 1 0 0 0 1 0 1 1 1 ...
K	1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 ...
C= M⊕K	1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 ...
K	1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 ...
M= C⊕K	0 1 1 1 0 0 1 0 0 0 1 0 1 1 1 ...

What do we mean by Security of a system?

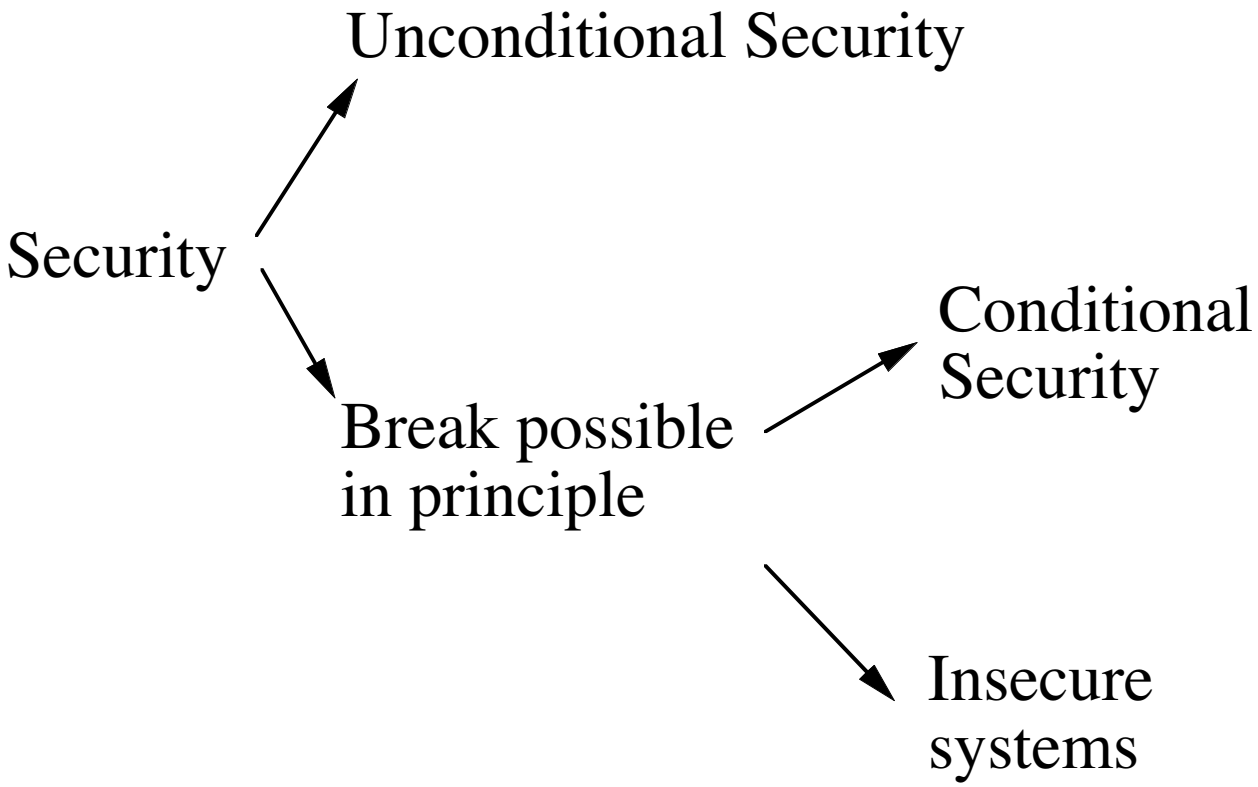
Model for Communikation (in case of Confidentiality)



Kerkhoff's Principle:

The algorithms G, E and D should be assumed to be known by the adversary. So security should be based solely on the fact that one of more keys are unknown to the attacker.

Classification of Security

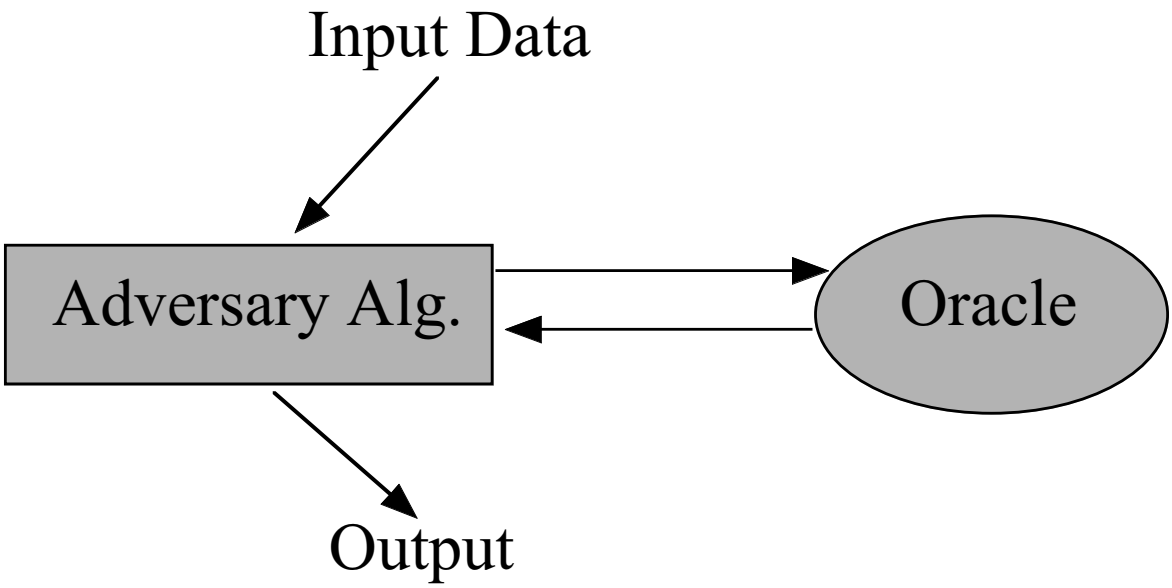


Unconditional Security:
System is secure, even if adversary has unbounded computing power.
Security measured via *Information Theory*.

Conditional Security:
System can be broken in principle, but this requires more computing power than a realistic adversary would have.
Security measured via *Complexity Theory*.

MODEL OF ATTACKS
- for conditional security

We think of the adversary as playing a game:



Input Data:

- whatever the adversary necessarily knows from the beginning, for instance the public key in case of a public key system, the distribution of plaintexts, etc.

Oracle

- models the information the adversary can obtain during an attack. Depending on the type of information the oracle provides, we get different types of attacks.

Output

- is whatever the adversary is trying to compute, he wins the game if he succeeds. The output could be the secret key, but could be something much less ambitous, say just partial information on some plaintext.

Types of Attacks (for Crypto Systems)

Ciphertext Only
Some distribution of plaintexts is given. The oracle provides encryptions of plaintexts under a fixed key.

Known Plaintext
Some distribution of plaintexts is given. The oracle provides on request a plaintext and its encryption under a fixed key.

Chosen Plaintext
The adversary can choose a plaintext, give it to the orcale, and the oracle will return its encryption under a fixed key.

Chosen Ciphertext
The adversary can choose a ciphertext, give it to the oracle, and the oracle will return the corresponding plaintext under a fixed key.

How to Build a Definition of Security

- specify an oracle (a type of attack)
- define what the adversary needs to do to win the game - a condition on his output.
- the system is secure under the definition, if any *efficient* adversary wins the game with only *negligible* probability.

A Standard Definition (for conventional encryption).

- no input data for adversary
- chosen plaintext attack, in the following sense:
  In case 0, when asked for encryption of message m, the oracle returns encryption of m under a fixed key
  that is randomly chosen initially; or
  in case 1, the oracle returns an encryption of a randomly chosen message, that is totally independent of m
- to win the game, the adversary must guess, whether he is in case 1 or case 0, so his output is just 1 bit.

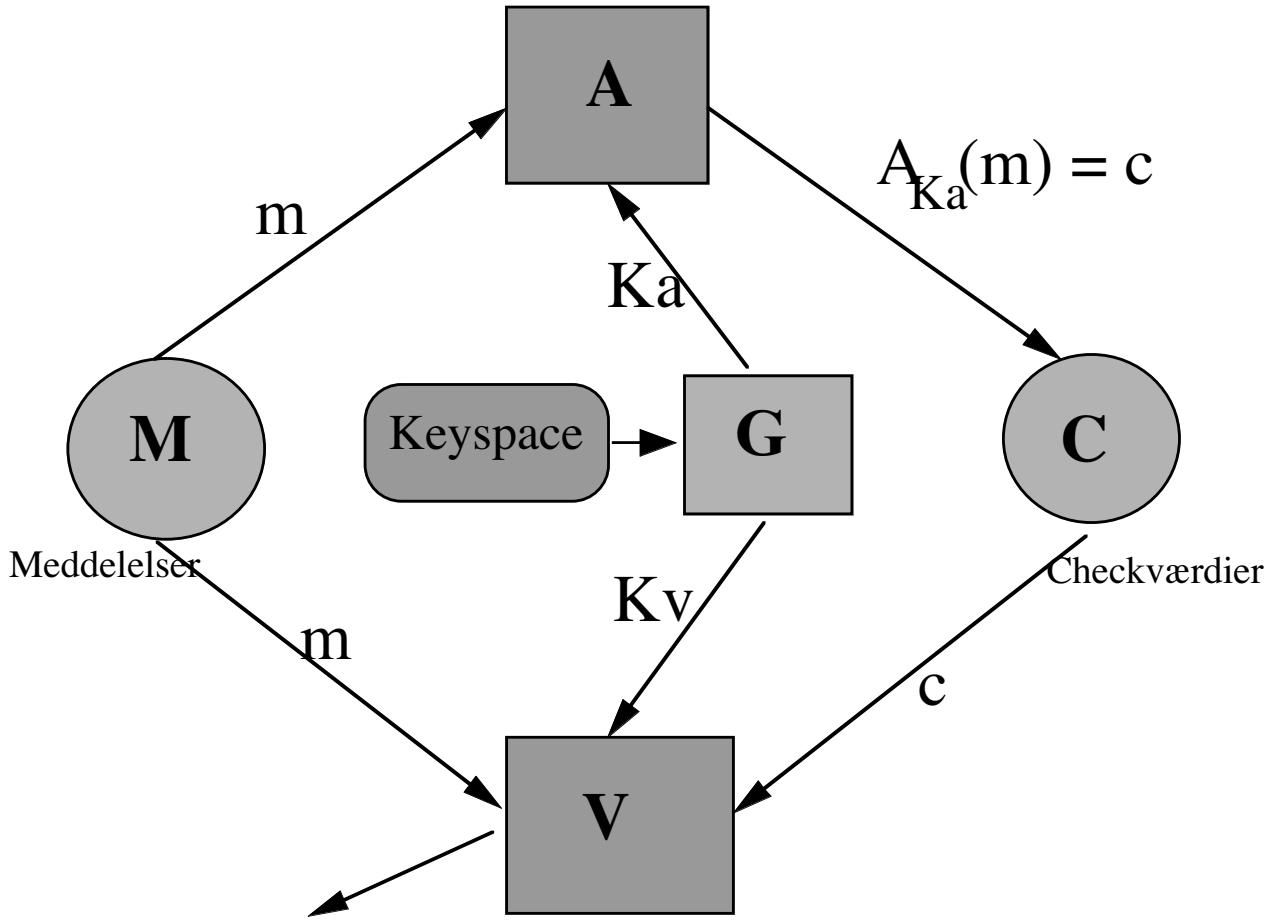
<u>The idea behind this:</u>
In case 1, the adversary gets completely useless data from the oracle. If he cannot even tell this apart from correct encryptions, he can do no damage in the real world (case 0) either.

P0 = probability that adversary A guesses “0” in case 0
P1 = probability that adversary A guesses “0” in case 1

Advantage of A is $Adv(A) = |P0 - P1|$

The cryptosystem is *(t,q,m,e)-secure* if any adversary A that runs in time t, makes <q quiries totalling <m bits, has an advantage $Adv(A) < e$.

Authentication System



$V_{Kv}(m,c) =$ *accept* el. *reject*

The system is given by a description of the algorithms A and C for generation and verification of check values, and the algorithm G for generating keys.

For any given key, A induces a mapping from M to C, and V a mapping from C,M to {*accept, reject* }.
For any (Ka, Kv) output by G, it must be the case that
Kv(m, Ka(m)) = *accept*

Conventionel (Classical System): Ka = Kv.
Here, Ka(m) is called a *Message Authentication Code* (MAC).

Public Key System: infeasible to compute Ka from Kv.
==> Kv can be *public*, simultaneously with Ka being *secret*.
Here, Ka(m) is called a *digital signature*.

Conventional Authentication Systems

- can only be used for weak authenticity: the same key is used by sender and receiver. So the receiver cannot convince anyone that he did not himself generate a pair of matching message and checkvalue.

Public Key Systems

- can be used for digital signatures (strong authenticity): only the sender knows Ka, so only the sender can generate a valid signature, which however anyone can check.

Types of Attacks on Authentication Systems

Known message attack:
- some distribution of messages is given. On request, the oracle outputs a pair (m,c) where m is a message and c is a check calue for m generated from some fixed key.

Chosen message attack:
The adversary can choose a message m any way he likes, give it to the oracle which will return a checkvalue c for m generated from some fixed key.

Goals for Attacks

Total Break
- Find the secret key Ka

Existential Forgery
- Find *any* message m for which you have not seen a checkvalue before, and compute a valid checkvalue c for m.

Standard definition of Secure Signature Schemes

For any adversary running in time polynomial in the length of the keys, the probability that he succeds in an existential forgery under a chosen message attack is negligible.

- the strongest possible definition.

How to get Authenticity from Integrity

Let us assume we have a secure authentication system

Problem:
The adversary can copy a correct pair (m,c) and send it again later (the replay problem).

Solution 1:
Message is expanded with a time stamp t, and check values are computed over both m og t, so ((m,t),c) is sent.
The receiver must verify c, and that t is not too far from his own system time.
Demands a certain synchronization between sender and receiver.

Solution 2:
Message is expanded with a sequence number n, and check values are computed over both m and n. So ((m,n),c) is sent.
The receiver must verify c, and that n equals his previous value of the counter +1 (and must then update the counter).

Demands storing a counter at sender and receiver and coordination of values if messages get lost..

Solution 3:
The receiver chooses a random number r, which is sent to the sender. Message is expanded with r, and check values are computed over both m and r. So ((m,r),c) is sent.
The receiver must verify c and compare r to the value he chose originally.

Demands an extra round of communication.