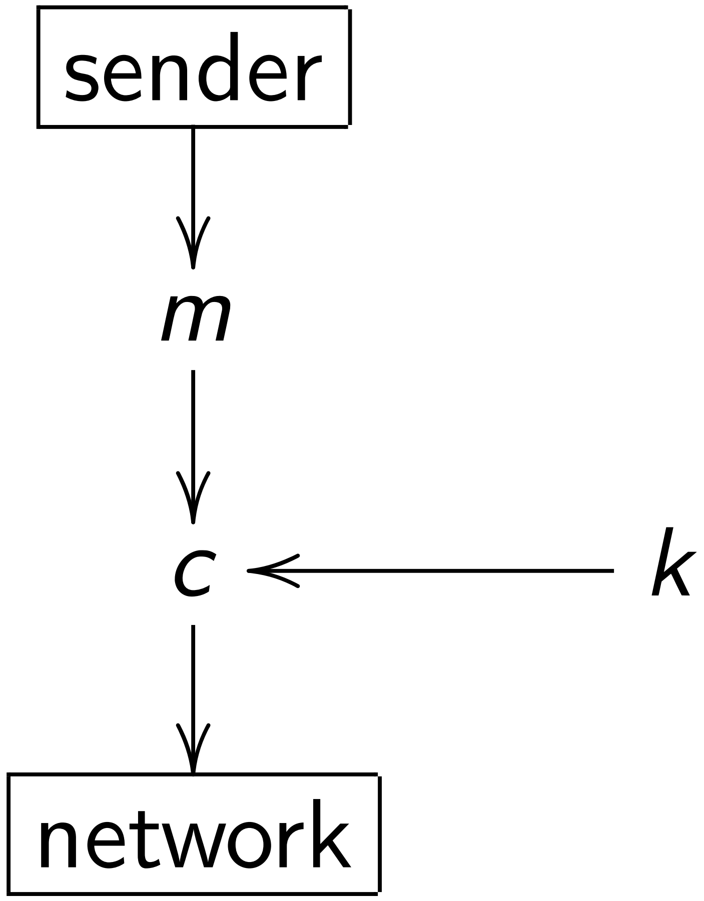
Goals of authenticated encryption

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

More details, credits:
competitions.cr.yp.to
/features.html

Encryption



k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

f

cated encryption

. Bernstein

ty of Illinois at Chicago &

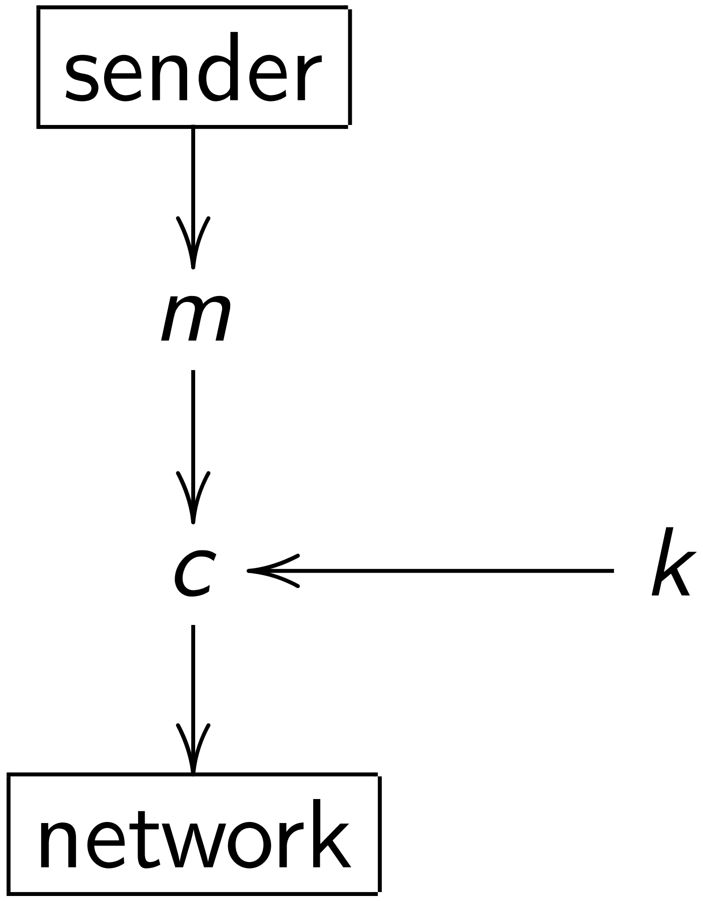
che Universiteit Eindhoven

tails, credits:

itions.cr.yp.to

res.html

Encryption



sender

m

c

k

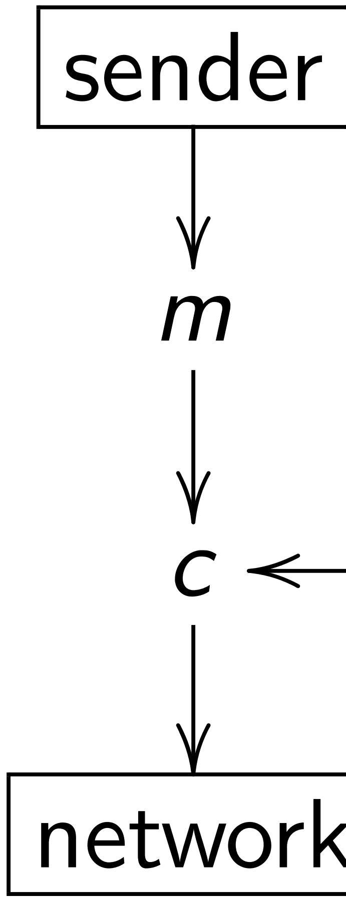
network

k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Authent



sender

m

c

network

k: secre

m: varia

c: varial

ryption

n

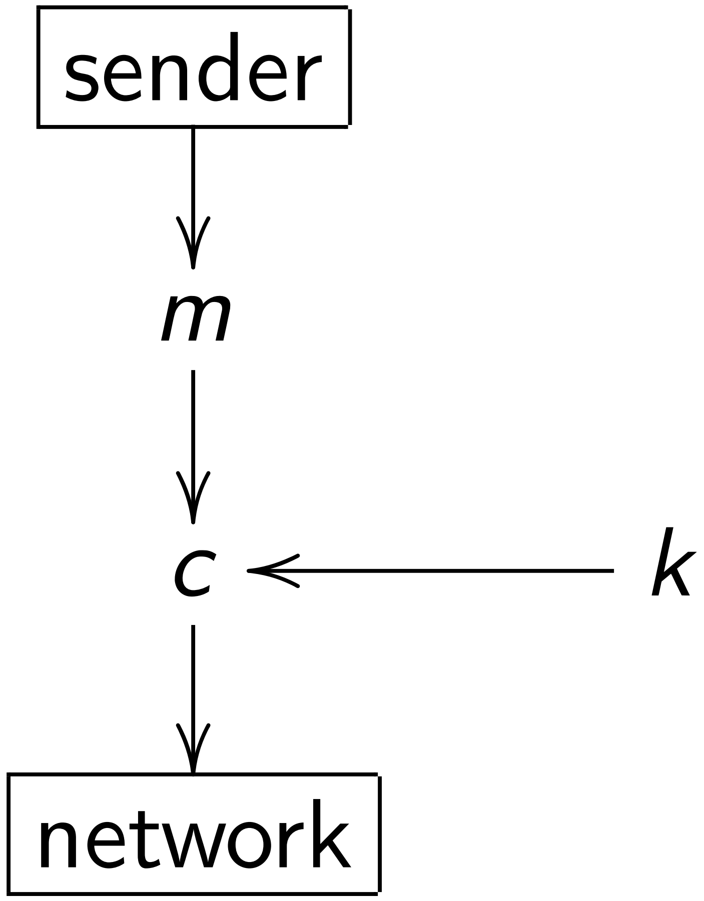
is at Chicago &

siteit Eindhoven

its:

r.yp.to

Encryption

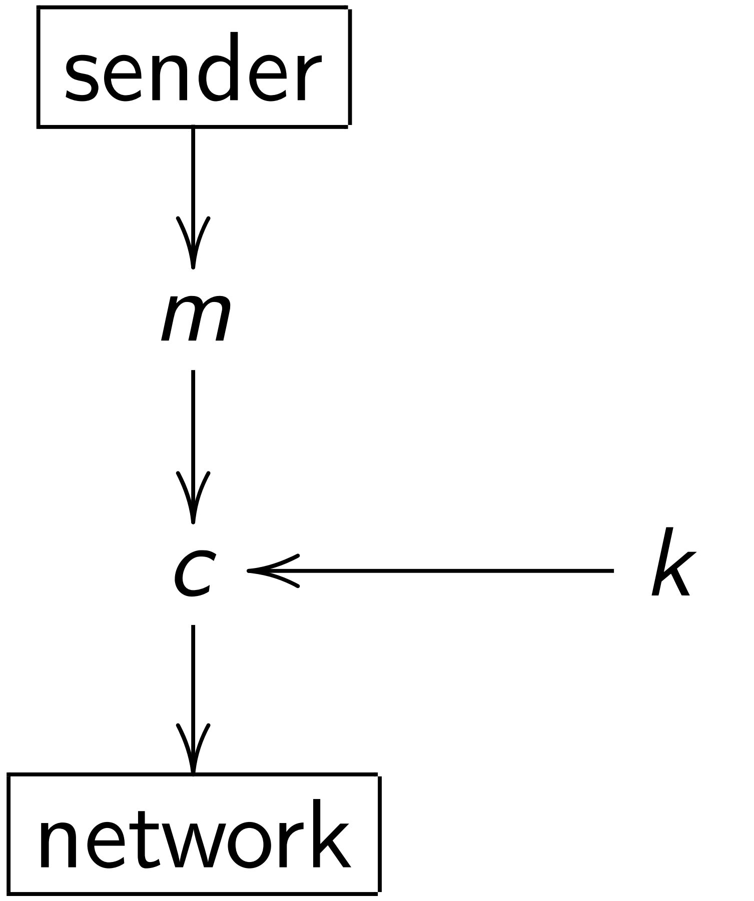


k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Authenticated enc



k: secret key.

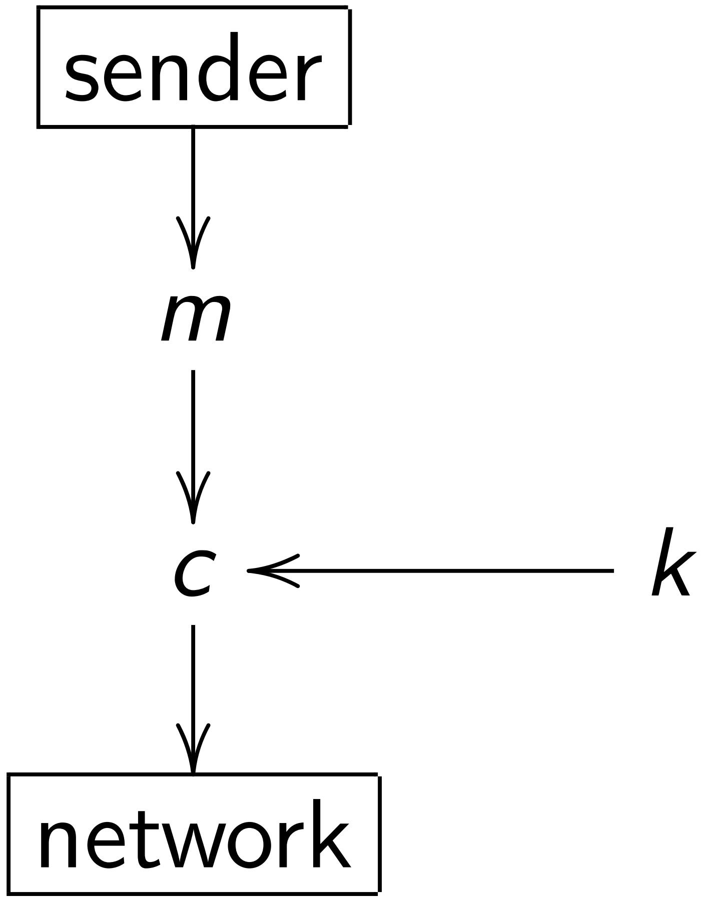
m: variable-length

c: variable-length

ago &

hoven

Encryption



sender

m

c

k

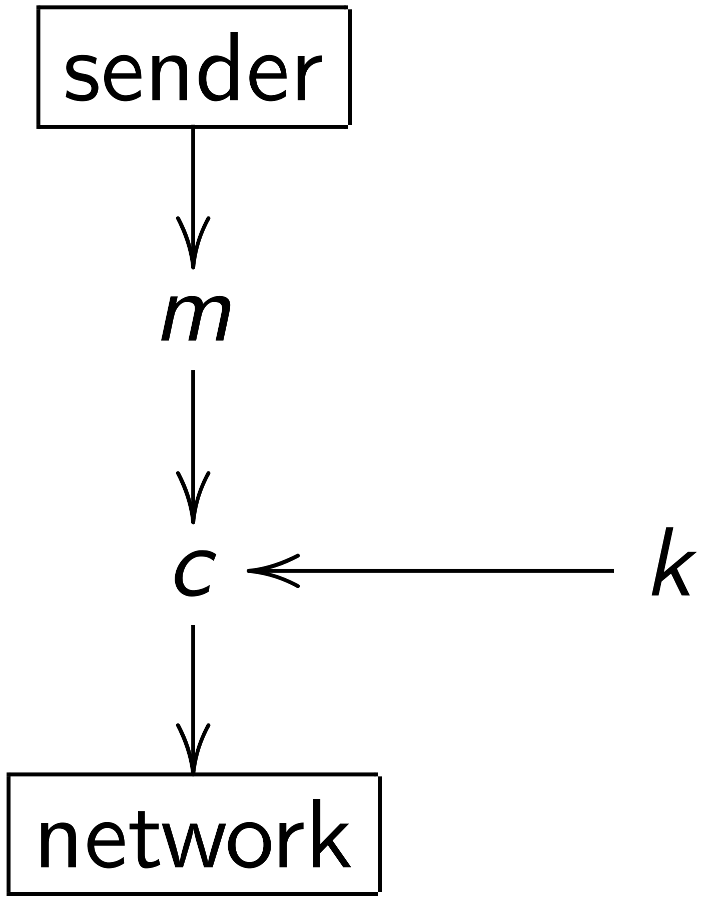
network

k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Authenticated encryption



sender

m

c

k

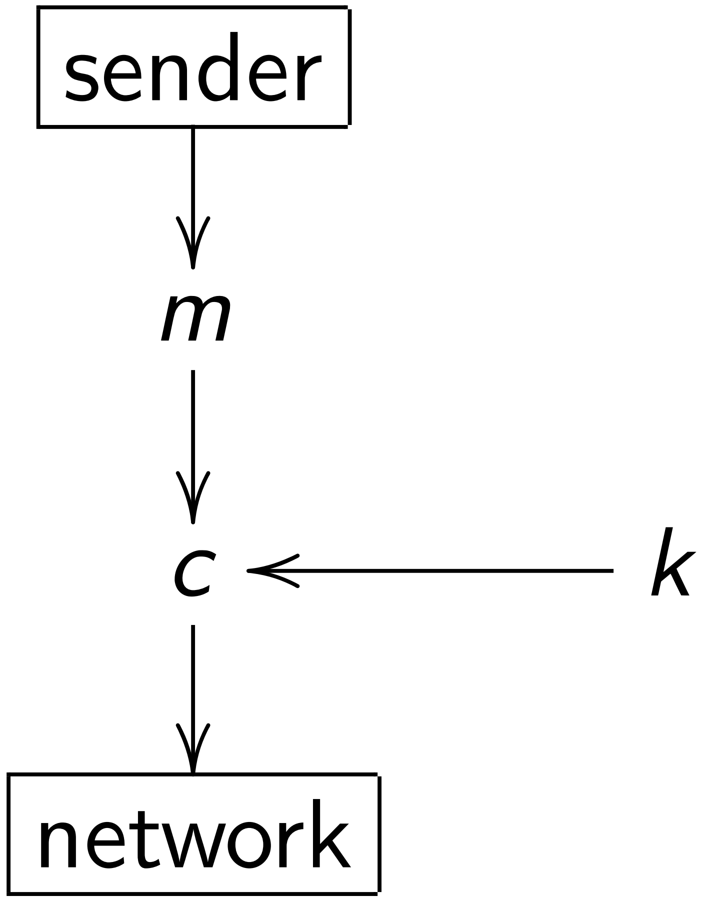
network

k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext

Encryption



sender

m

c

k

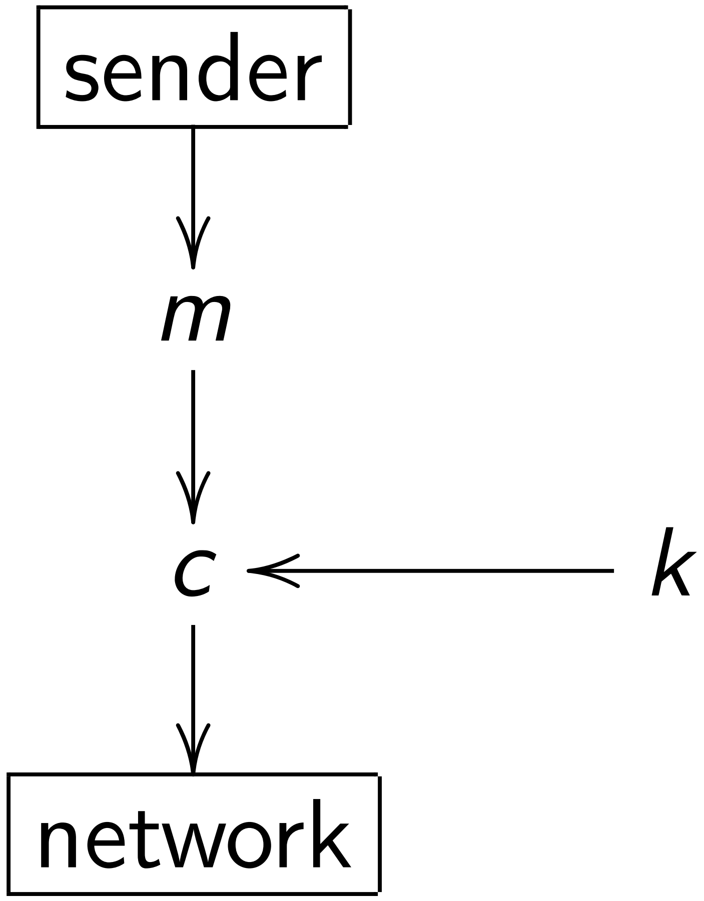
network

k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Authenticated encryption



sender

m

c

k

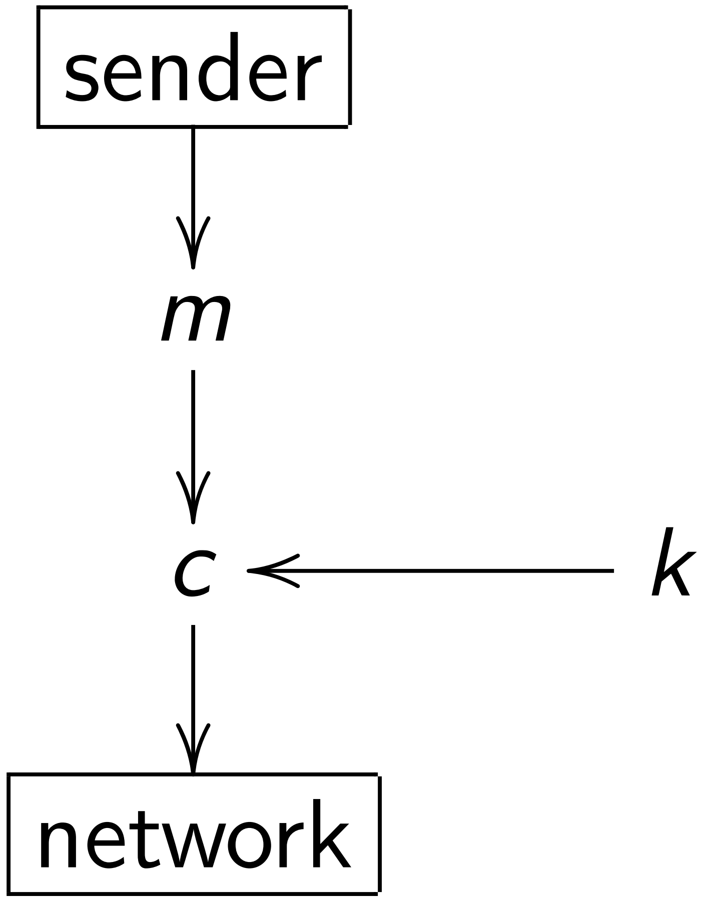
network

k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Encryption



sender

m

c

k

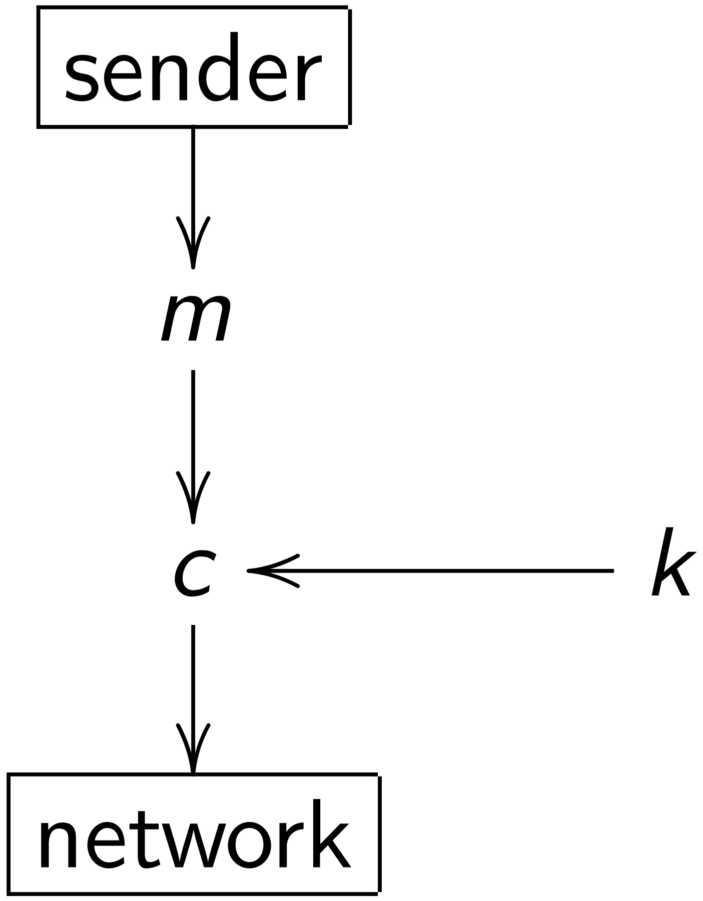
network

k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Authenticated encryption



sender

m

c

k

network

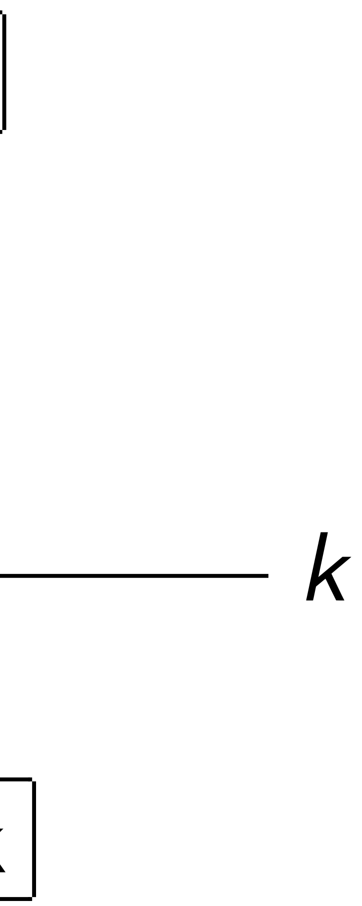
k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Same picture! But now c is slightly longer than m: includes an “authentication tag”.

on



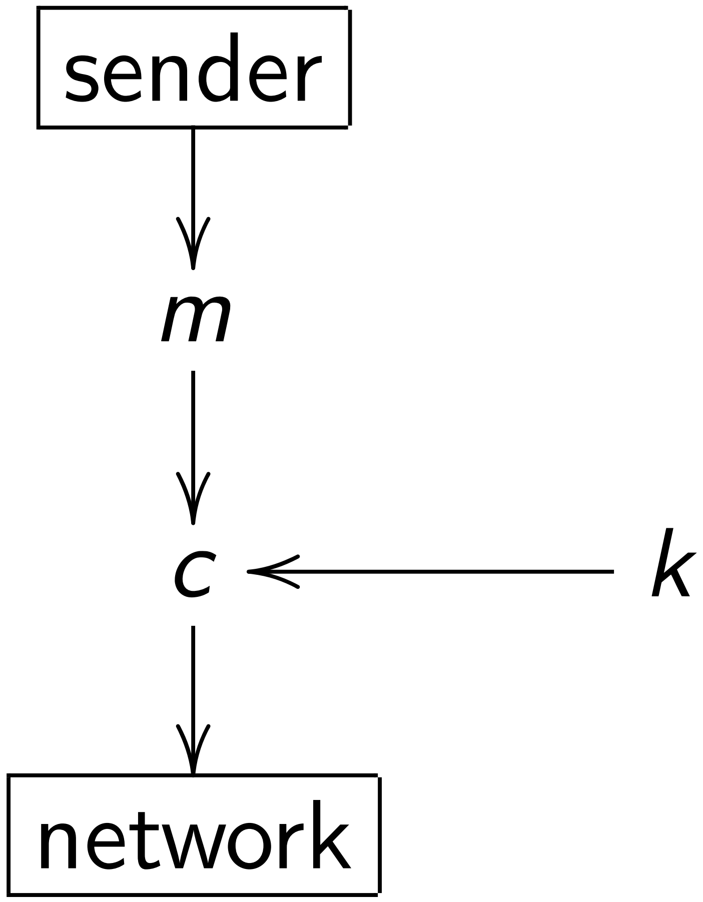
k

t key.

able-length plaintext.

ble-length ciphertext.

Authenticated encryption



sender

m

c

k

network

k: secret key.

m: variable-length plaintext.

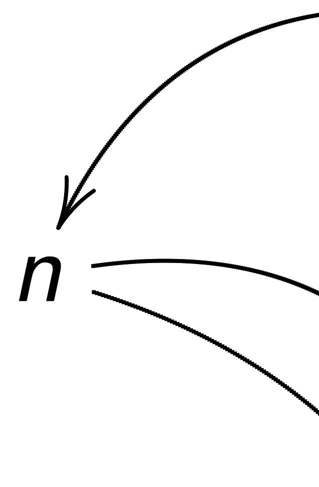
c: variable-length ciphertext.

Same picture! But now

c is slightly longer than m:

includes an “authentication tag”.

Message



n

k: secre

n: publi

m: varia

c: varial

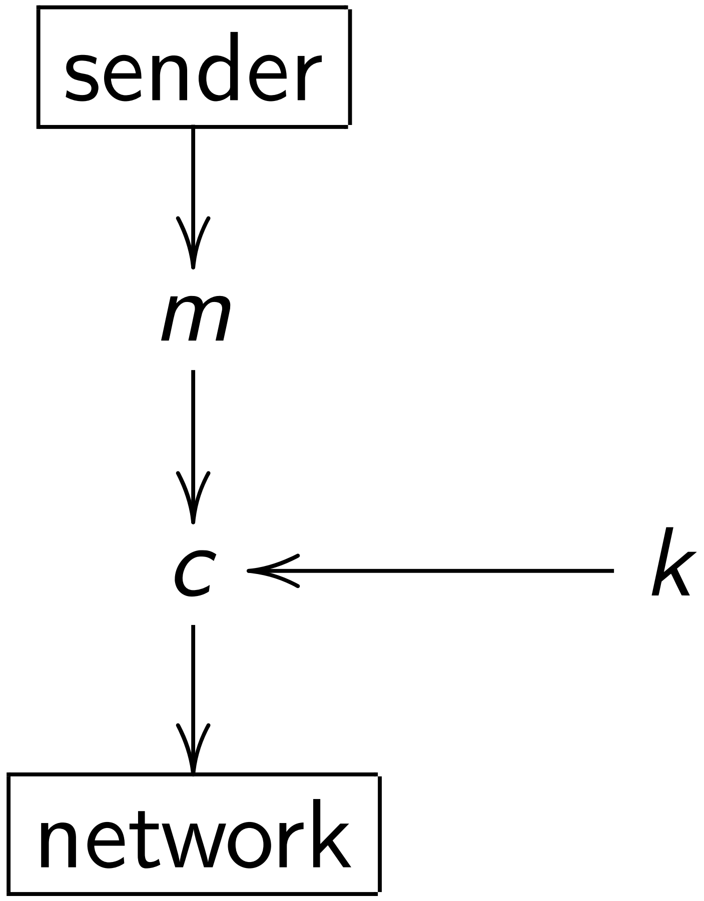
Changes

hide rep

plaintext.

ciphertext.

Authenticated encryption



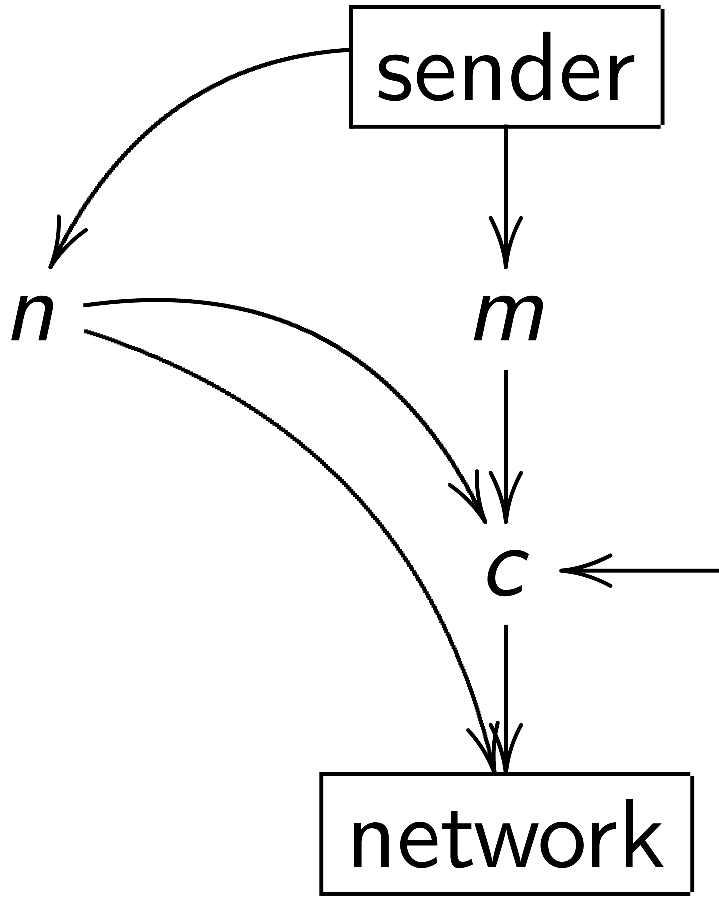
k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Same picture! But now
c is slightly longer than m:
includes an “authentication tag”.

Message numbers



k: secret key.

n: public message

m: variable-length

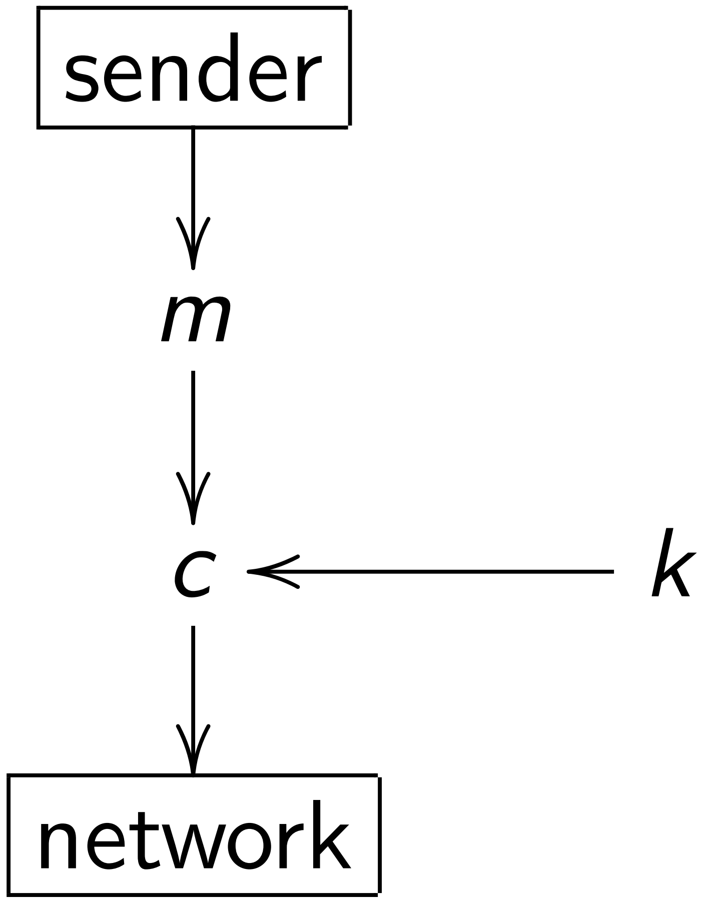
c: variable-length

Changes in messag
hide repetitions of

.

t.

Authenticated encryption



sender

m

c

k

network

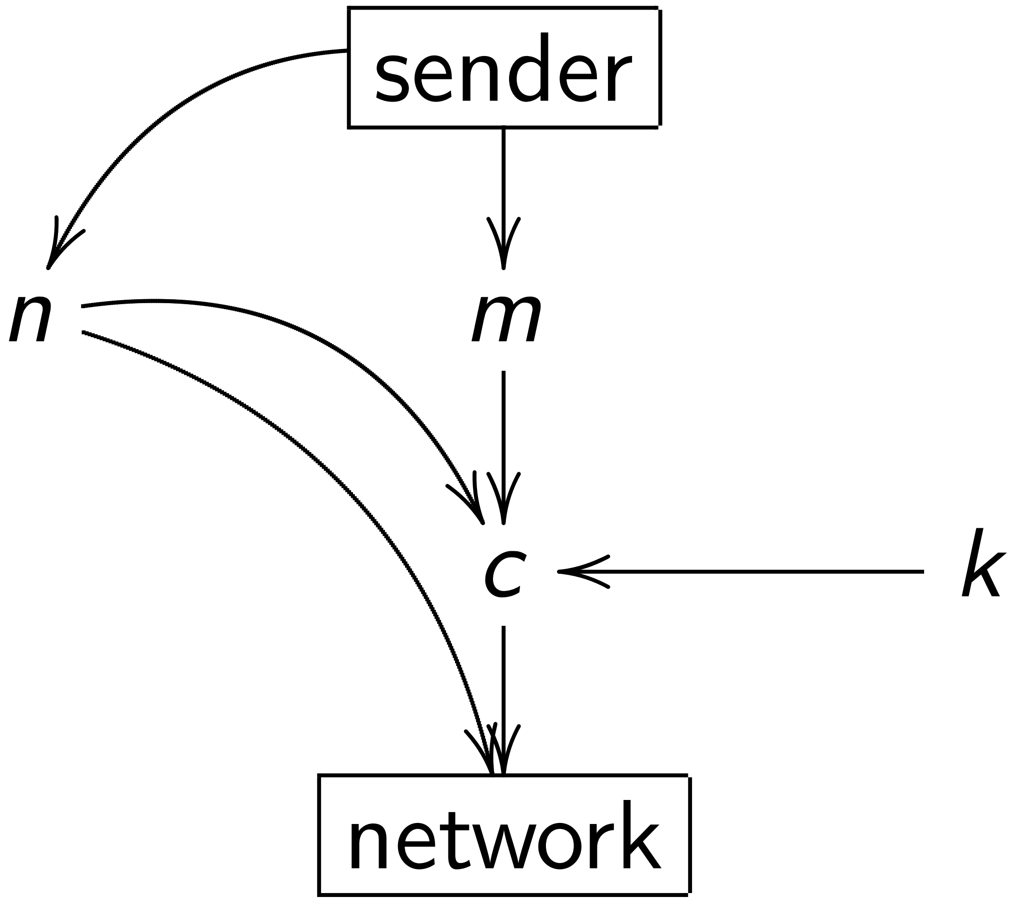
k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Same picture! But now c is slightly longer than m: includes an “authentication tag”.

Message numbers



sender

n

m

c

k

network

k: secret key.

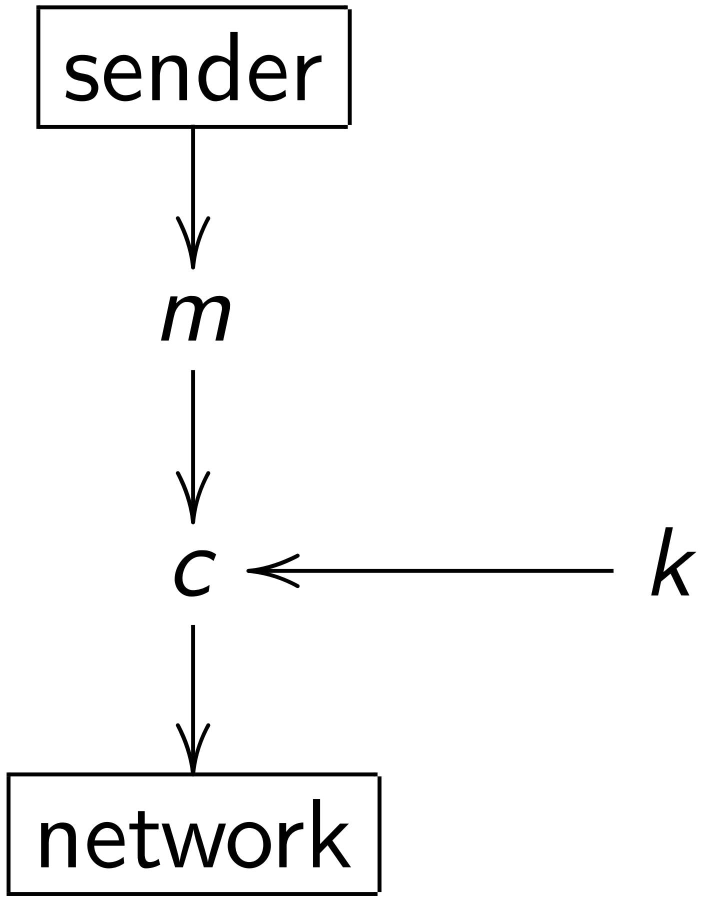
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext

Changes in message number hide repetitions of plaintext.

Authenticated encryption


sender

m

c

k

network

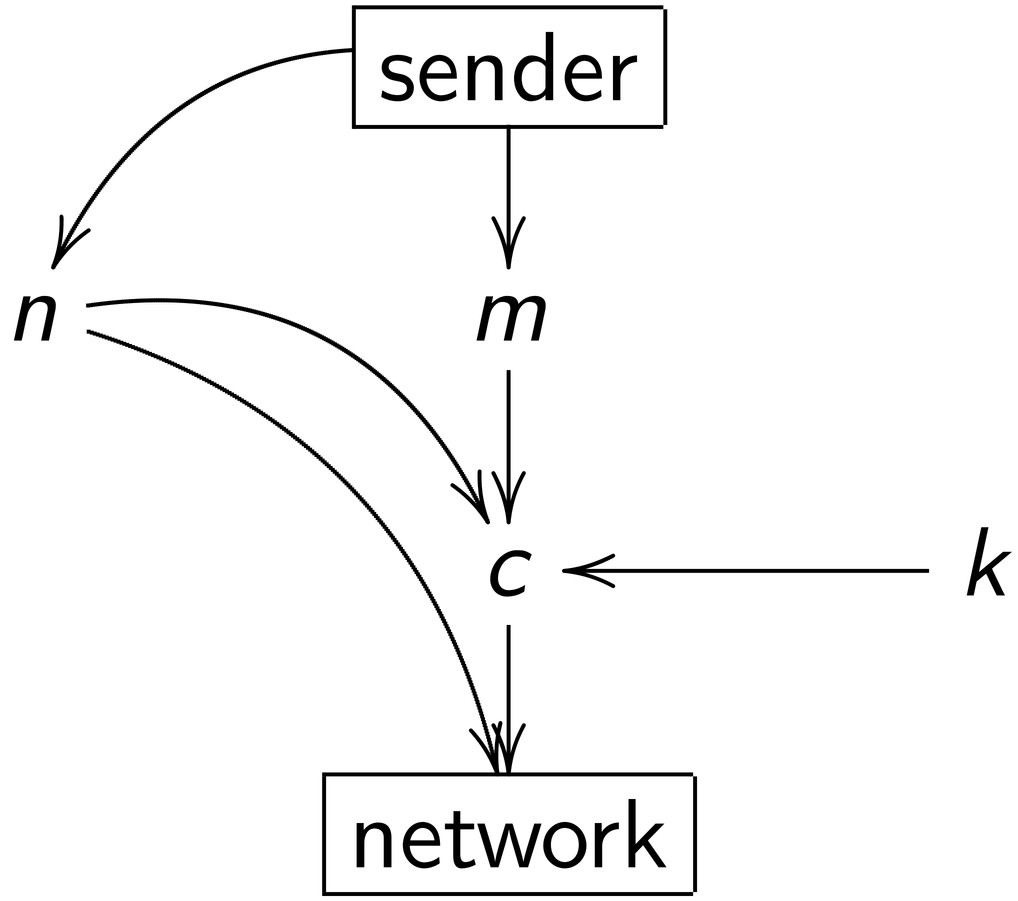
k: secret key.

m: variable-length plaintext.

c: variable-length ciphertext.

Same picture! But now c is slightly longer than m: includes an “authentication tag”.

Message numbers


sender

n

m

c

k

network

k: secret key.

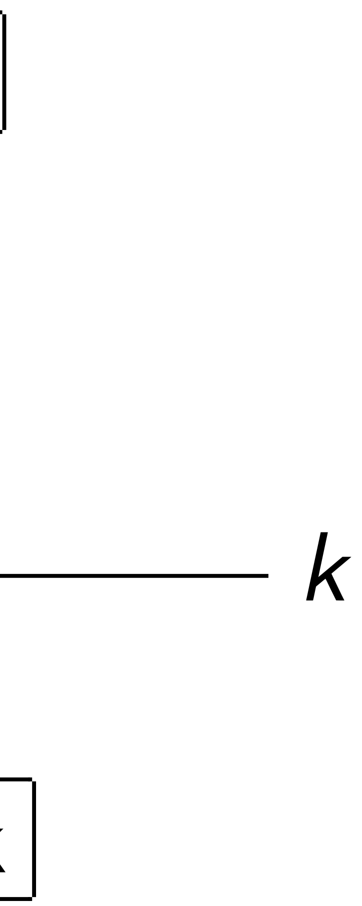
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext.

Changes in message number hide repetitions of plaintext.

icated encryption



t key.

able-length plaintext.

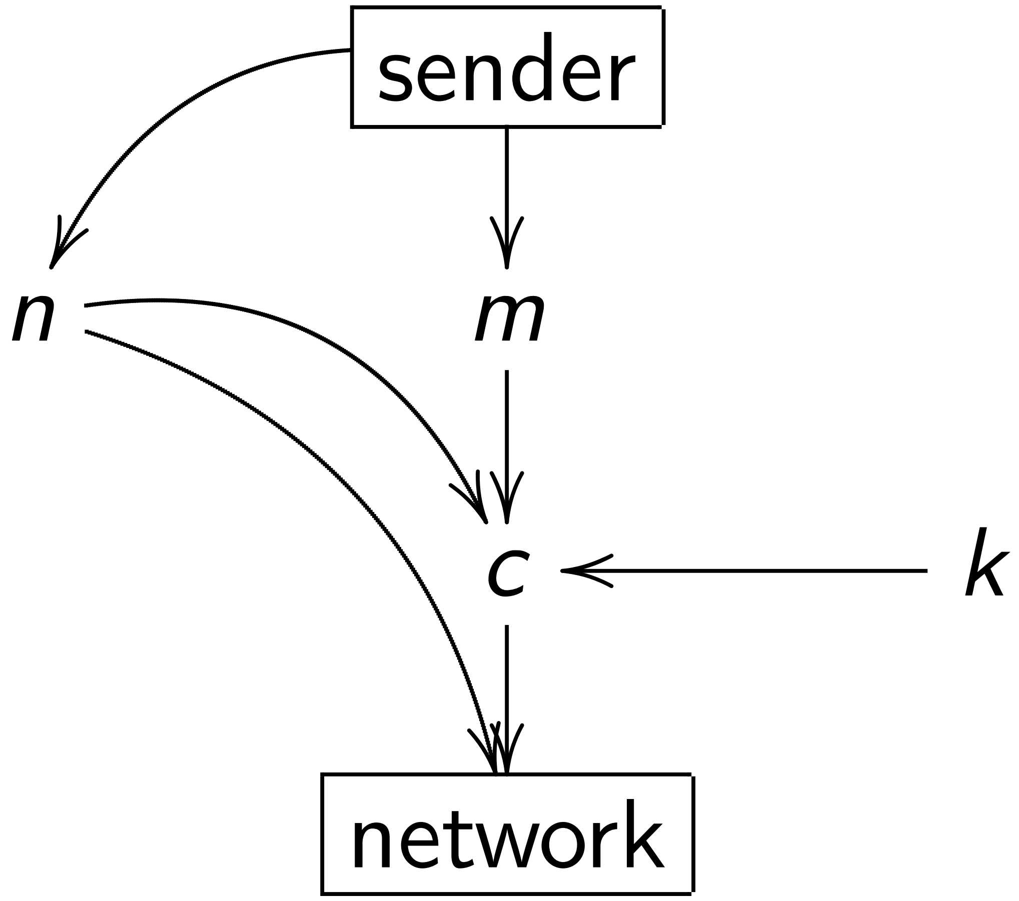
ble-length ciphertext.

cture! But now

ntly longer than m:

an “authentication tag”.

Message numbers



k: secret key.

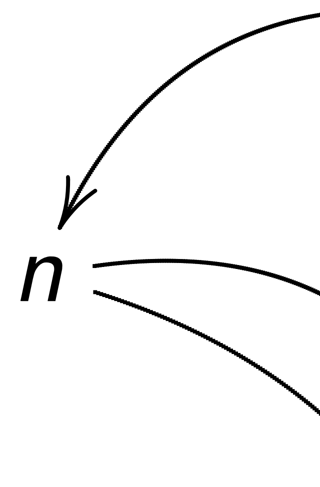
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext.

Changes in message number hide repetitions of plaintext.

Associat



k: secre

n: publi

a: variab

m: varia

c: varial

ryption

plaintext.

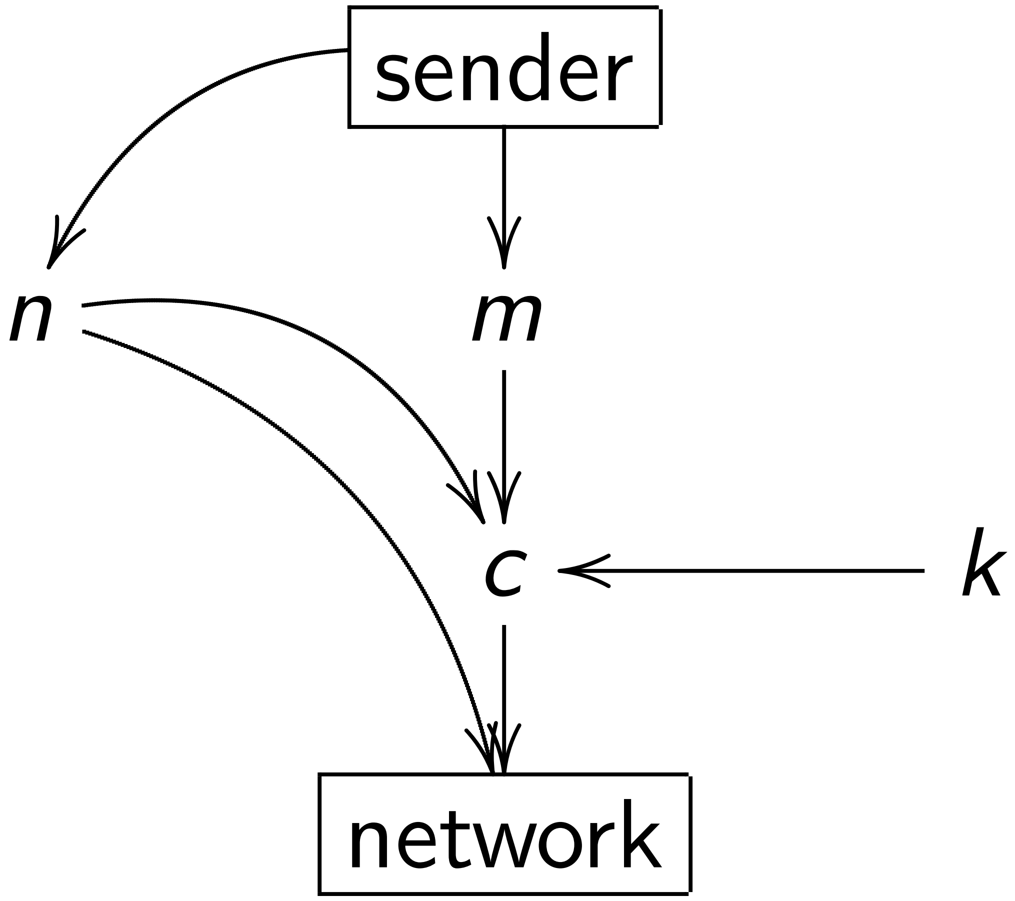
ciphertext.

t now

than m:

entication tag”.

Message numbers



k: secret key.

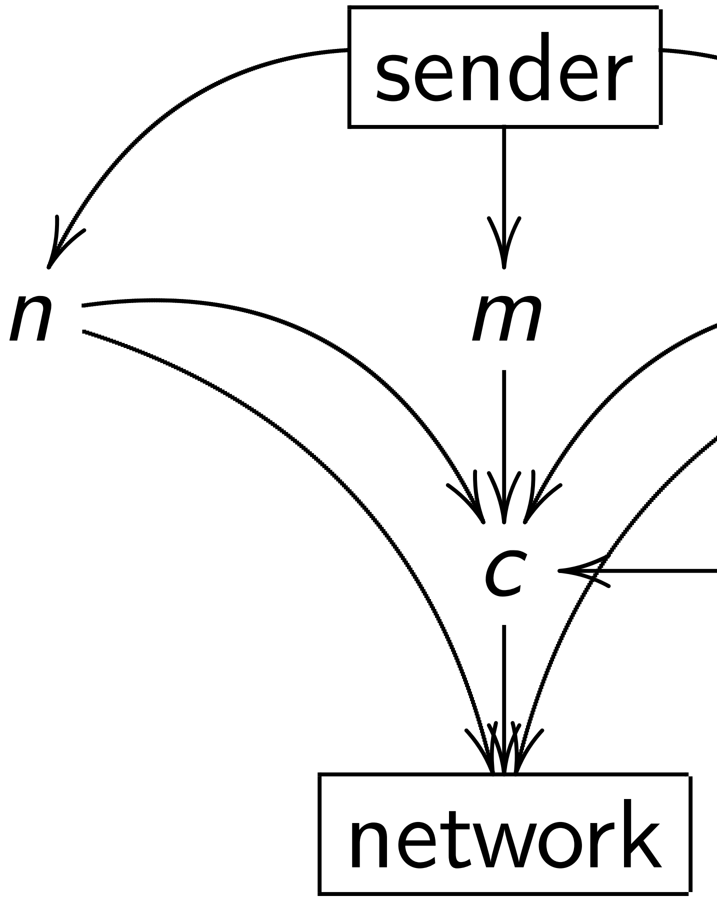
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext.

Changes in message number hide repetitions of plaintext.

Associated data



k: secret key.

n: public message

a: variable-length

m: variable-length

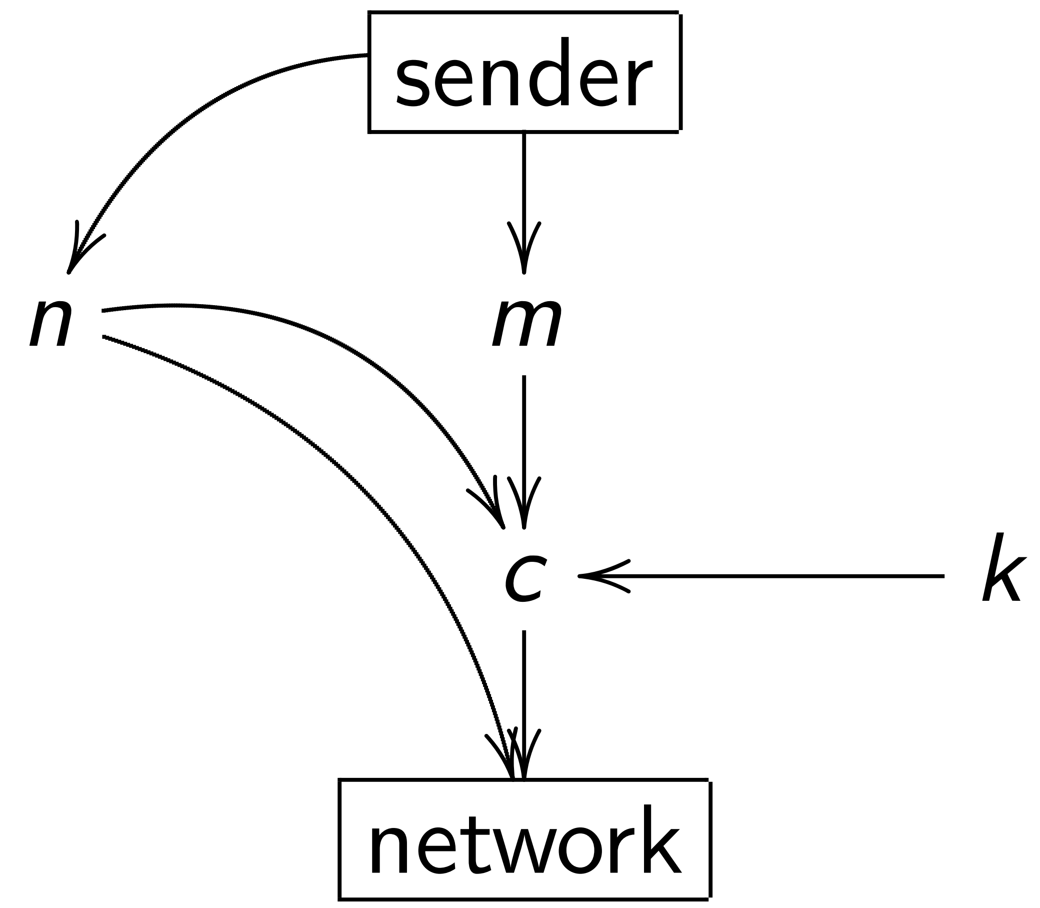
c: variable-length

.

t.

tag”.

Message numbers


sender

n m

c k

network

k: secret key.

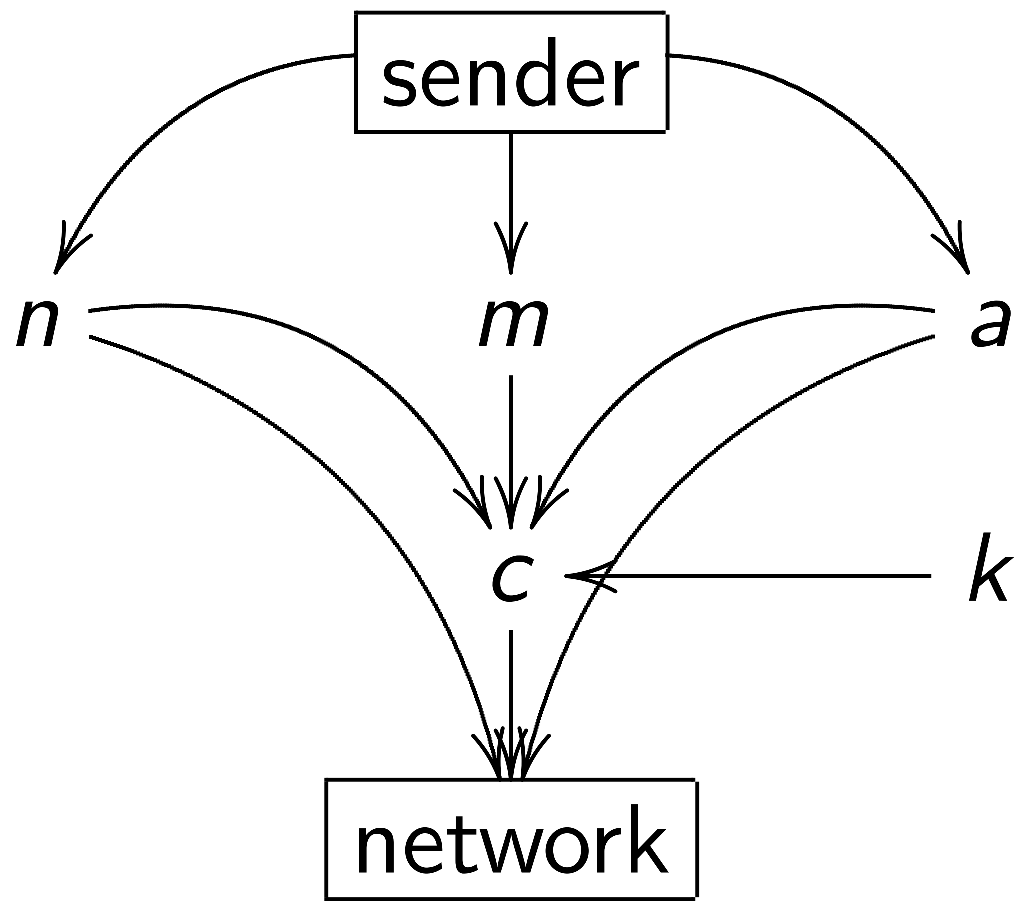
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext.

Changes in message number hide repetitions of plaintext.

Associated data


sender

n m a

c k

network

k: secret key.

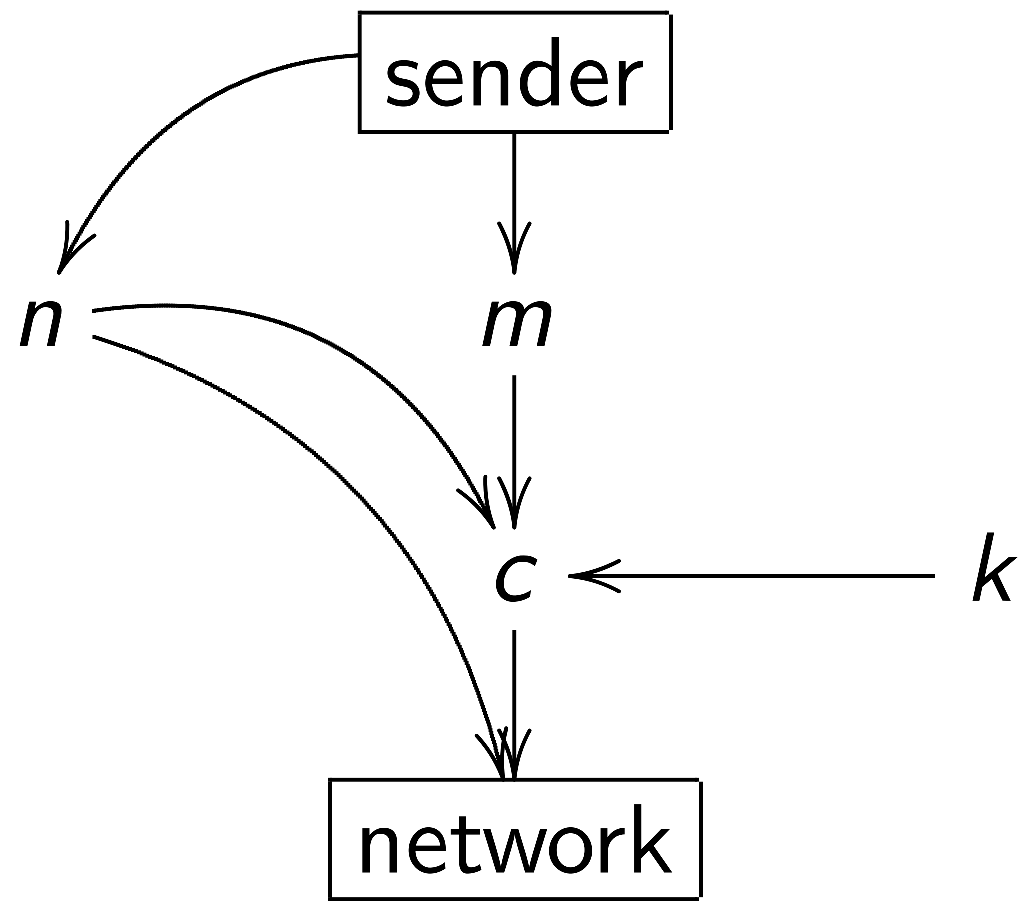
n: public message number.

a: variable-length associate

m: variable-length plaintext.

c: variable-length ciphertext

Message numbers



k: secret key.

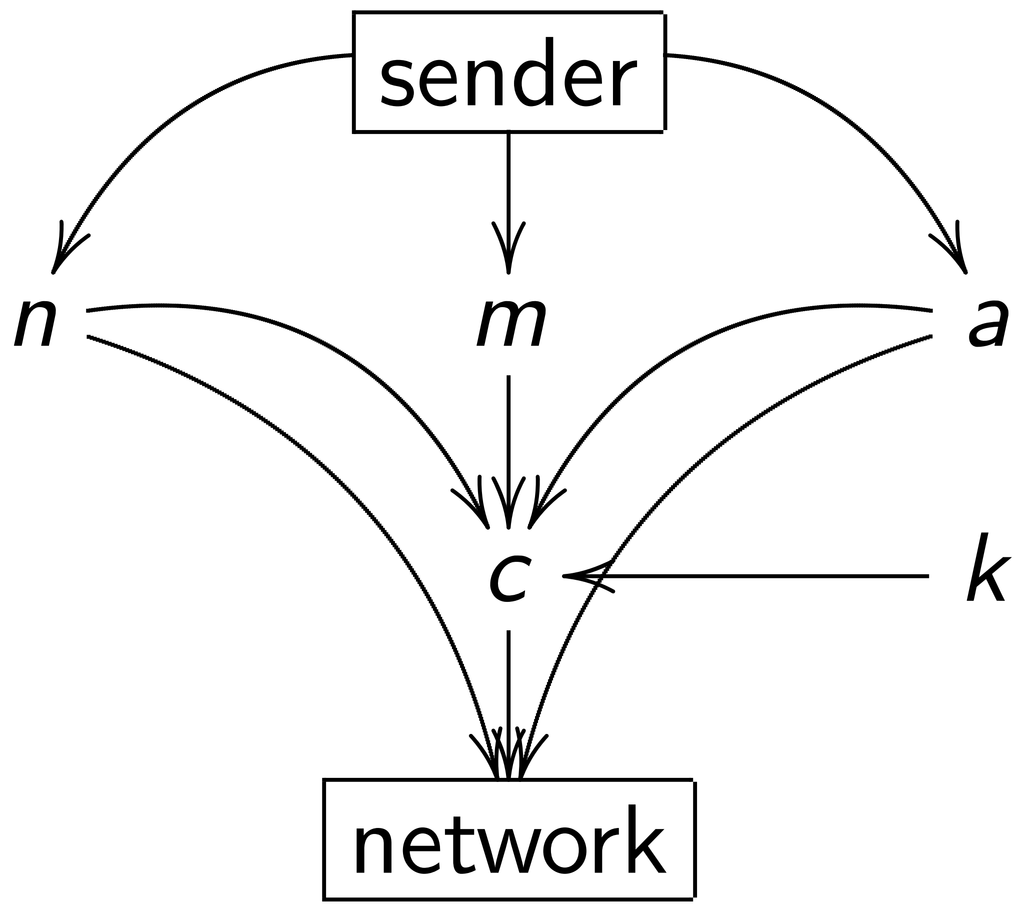
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext.

Changes in message number hide repetitions of plaintext.

Associated data



k: secret key.

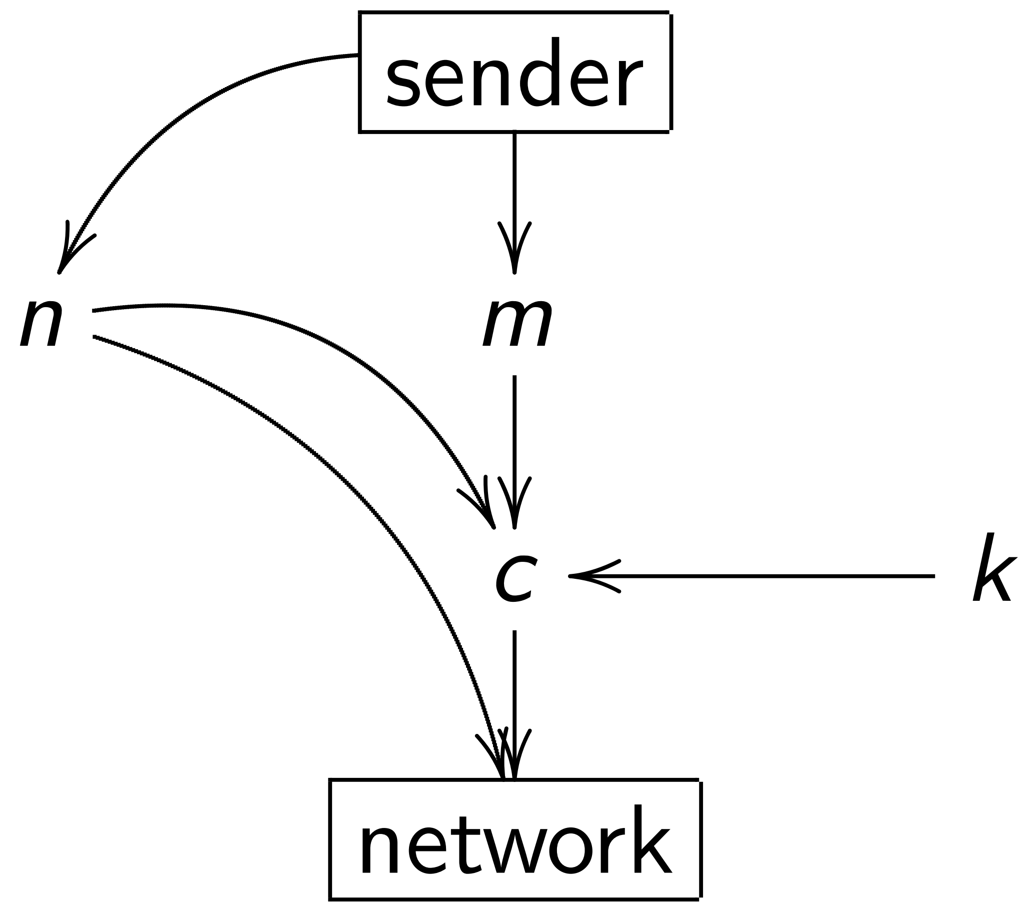
n: public message number.

a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

Message numbers



k: secret key.

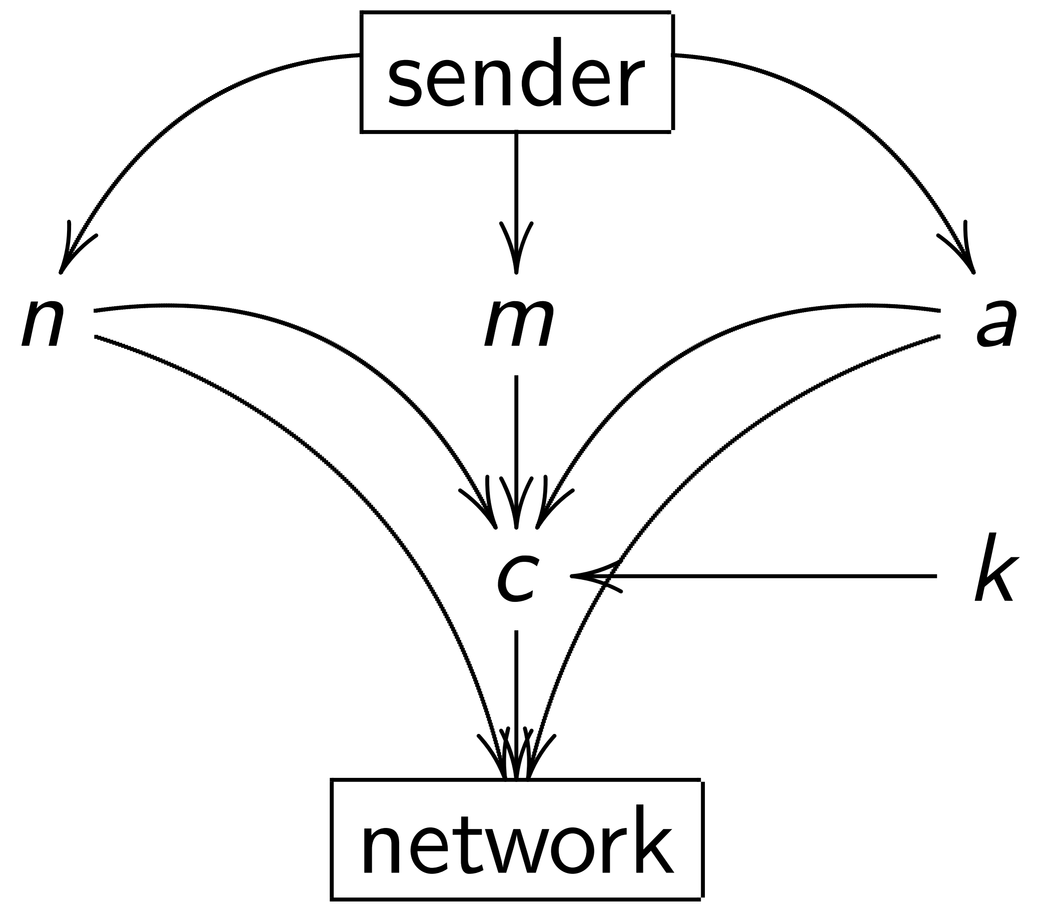
n: public message number.

m: variable-length plaintext.

c: variable-length ciphertext.

Changes in message number hide repetitions of plaintext.

Associated data



k: secret key.

n: public message number.

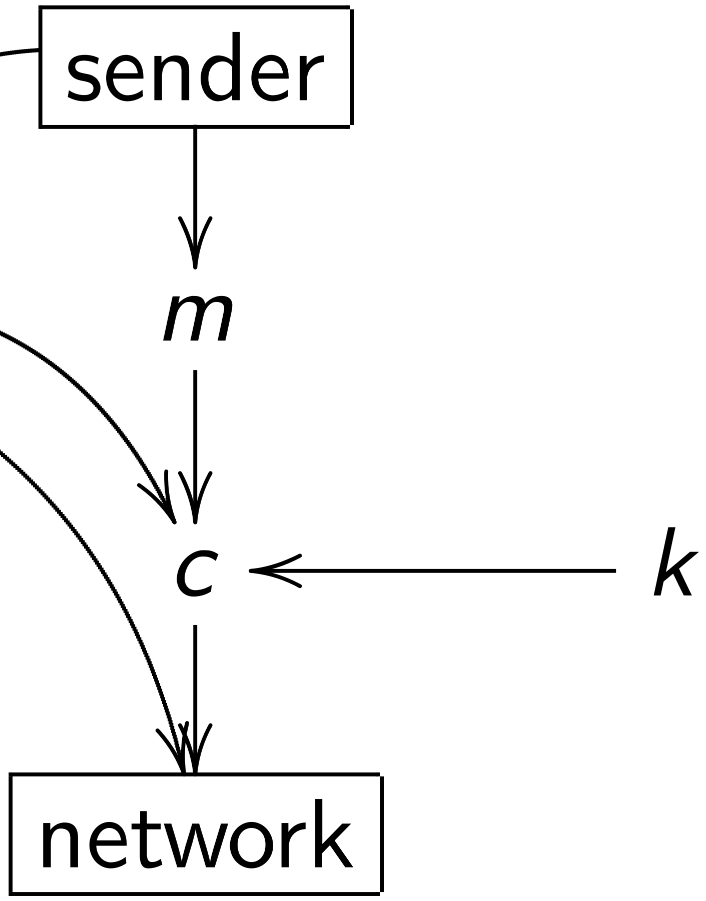
a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

No problem repeating a.

e numbers



t key.

c message number.

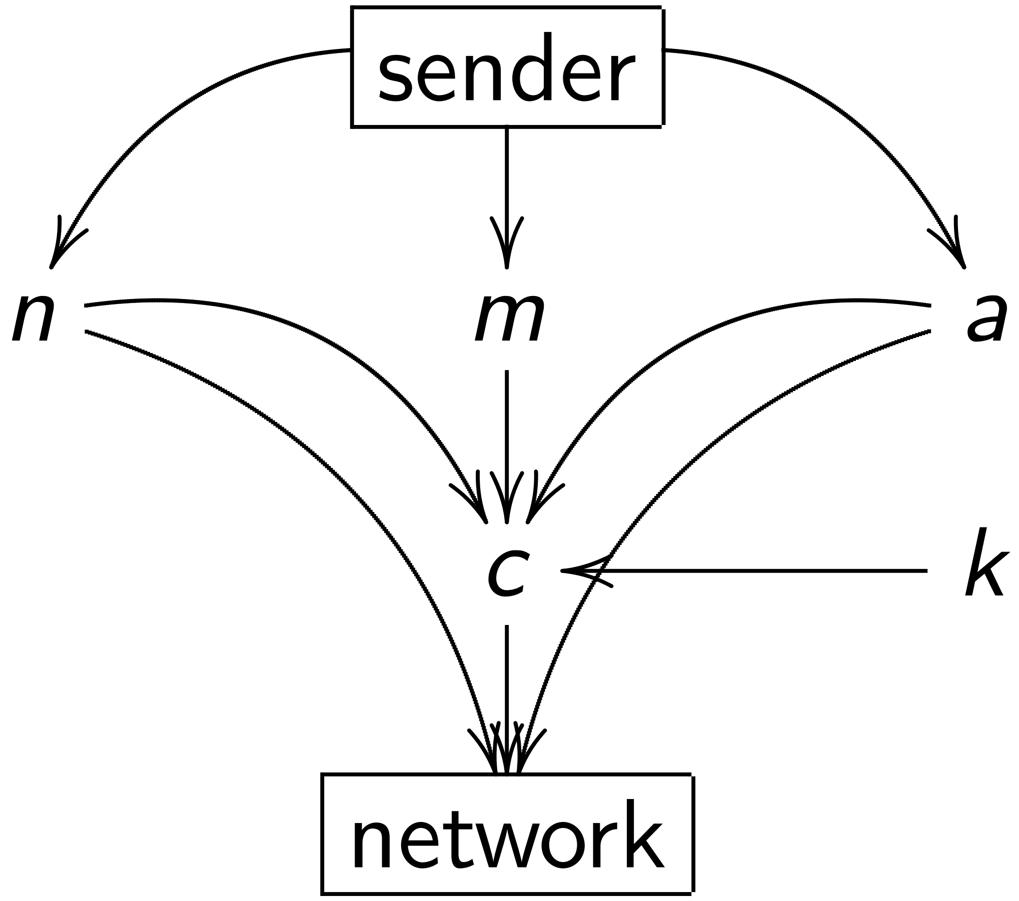
able-length plaintext.

ble-length ciphertext.

s in message number

etitions of plaintext.

Associated data



k: secret key.

n: public message number.

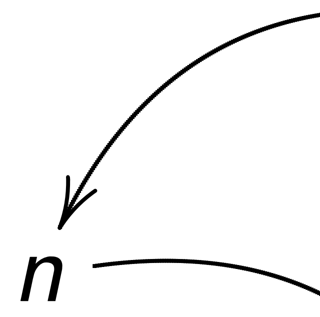
a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

No problem repeating *a*.

Secret m



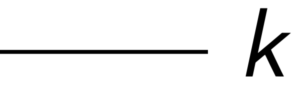
k: secre

n: secre

a: variab

m: varia

c: varial



number.

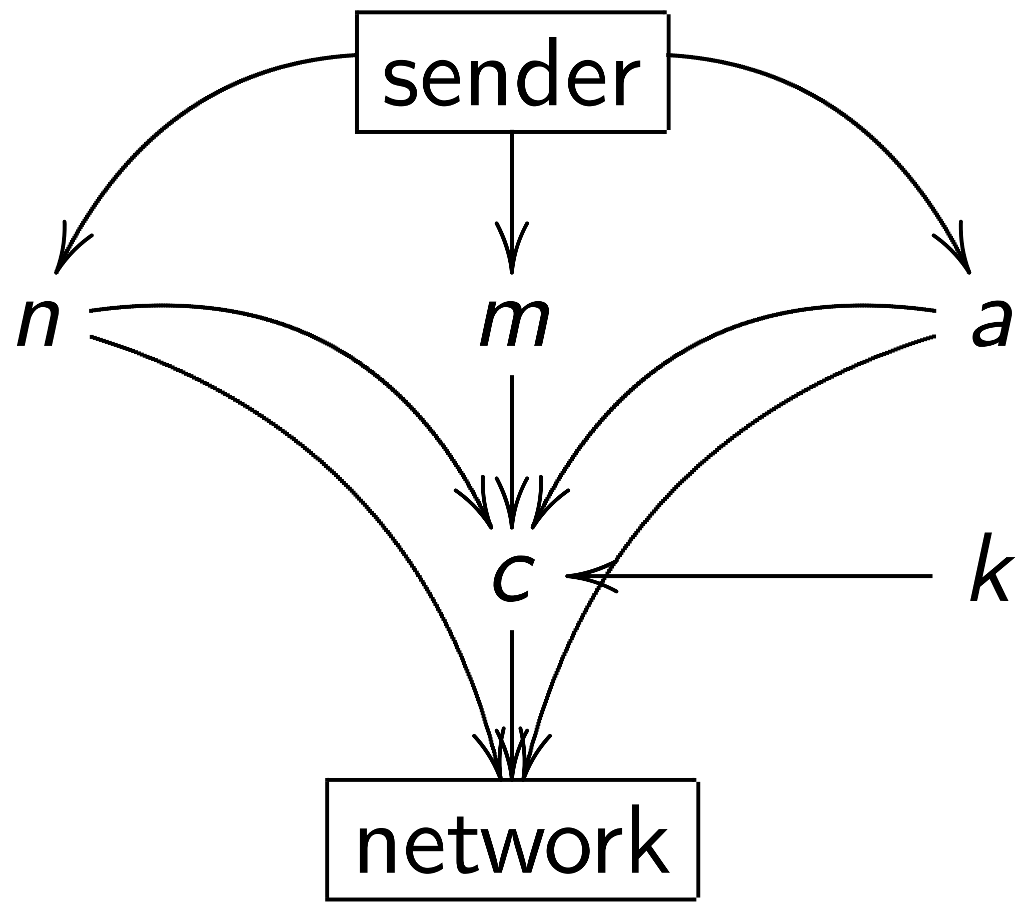
plaintext.

ciphertext.

ge number

plaintext.

Associated data



k: secret key.

n: public message number.

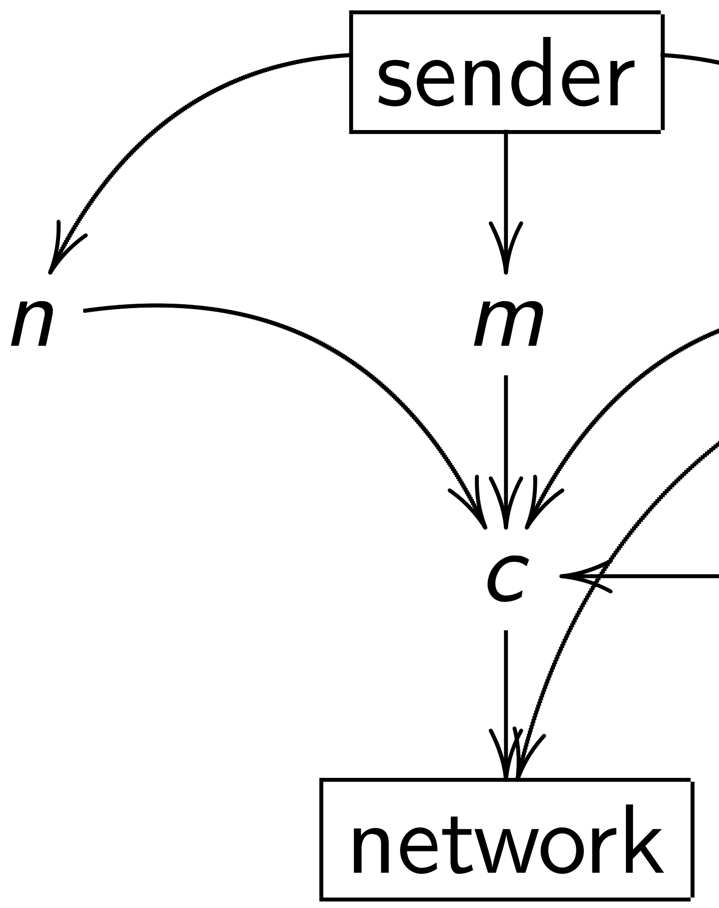
a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

No problem repeating a.

Secret message nu



k: secret key.

n: secret message

a: variable-length

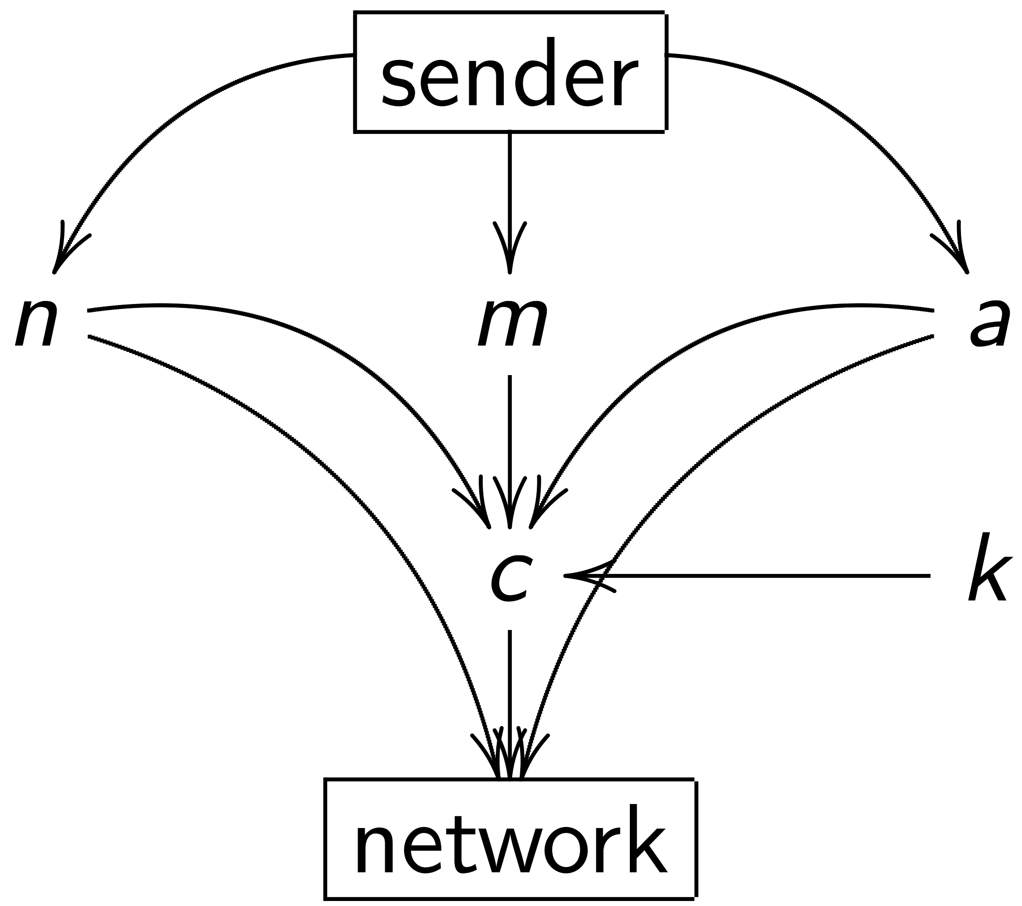
m: variable-length

c: variable-length

.

t.

Associated data



k: secret key.

n: public message number.

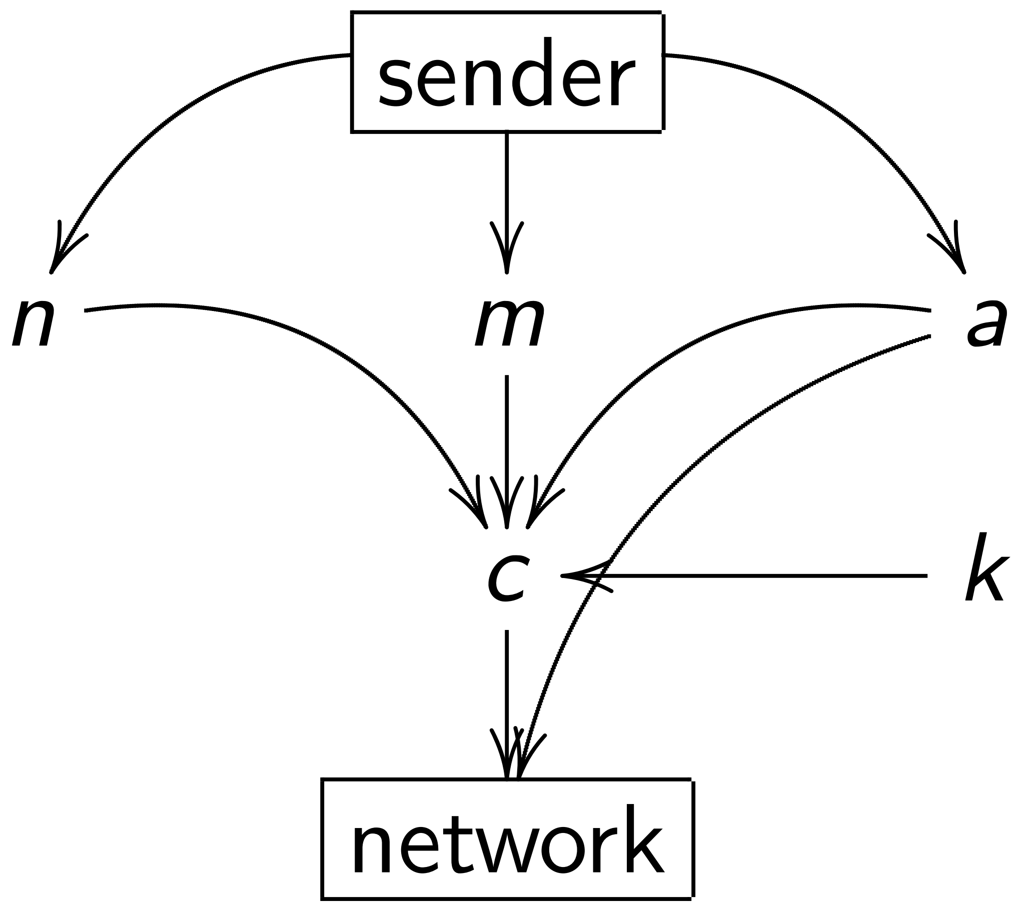
a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

No problem repeating a.

Secret message numbers



k: secret key.

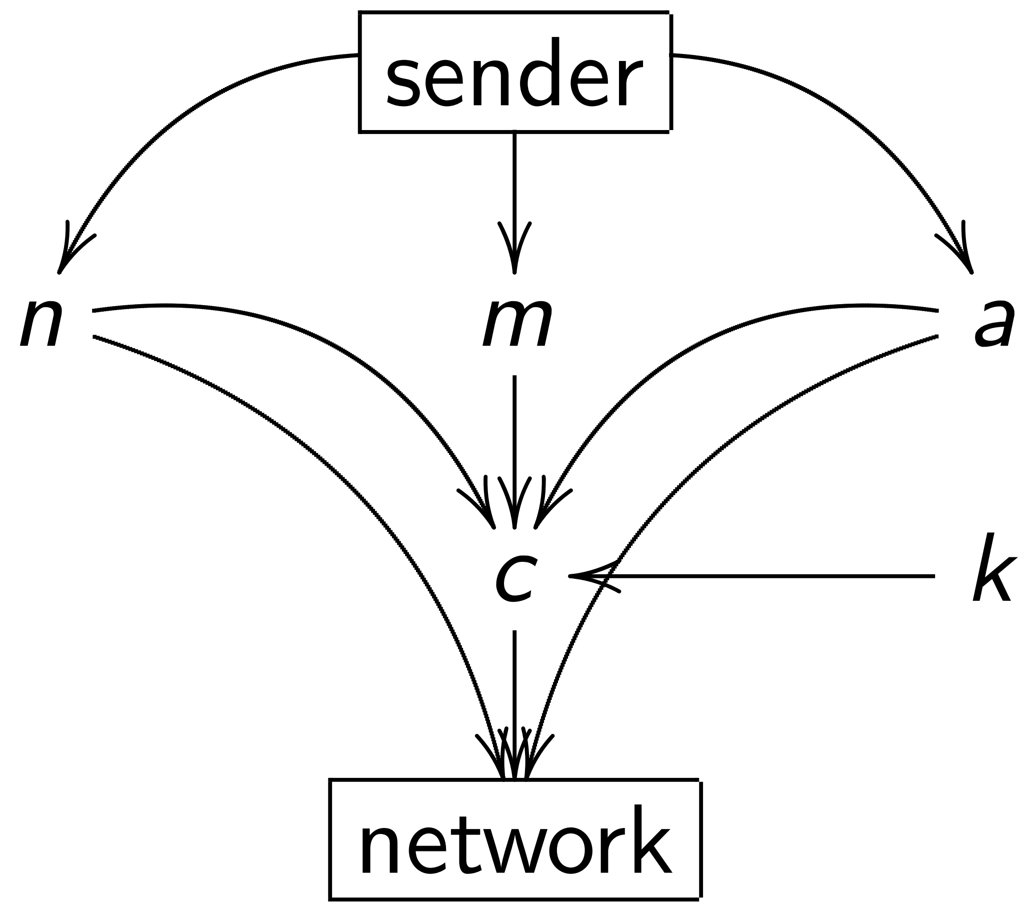
n: secret message number.

a: variable-length associatec

m: variable-length plaintext.

c: variable-length ciphertext

Associated data



k: secret key.

n: public message number.

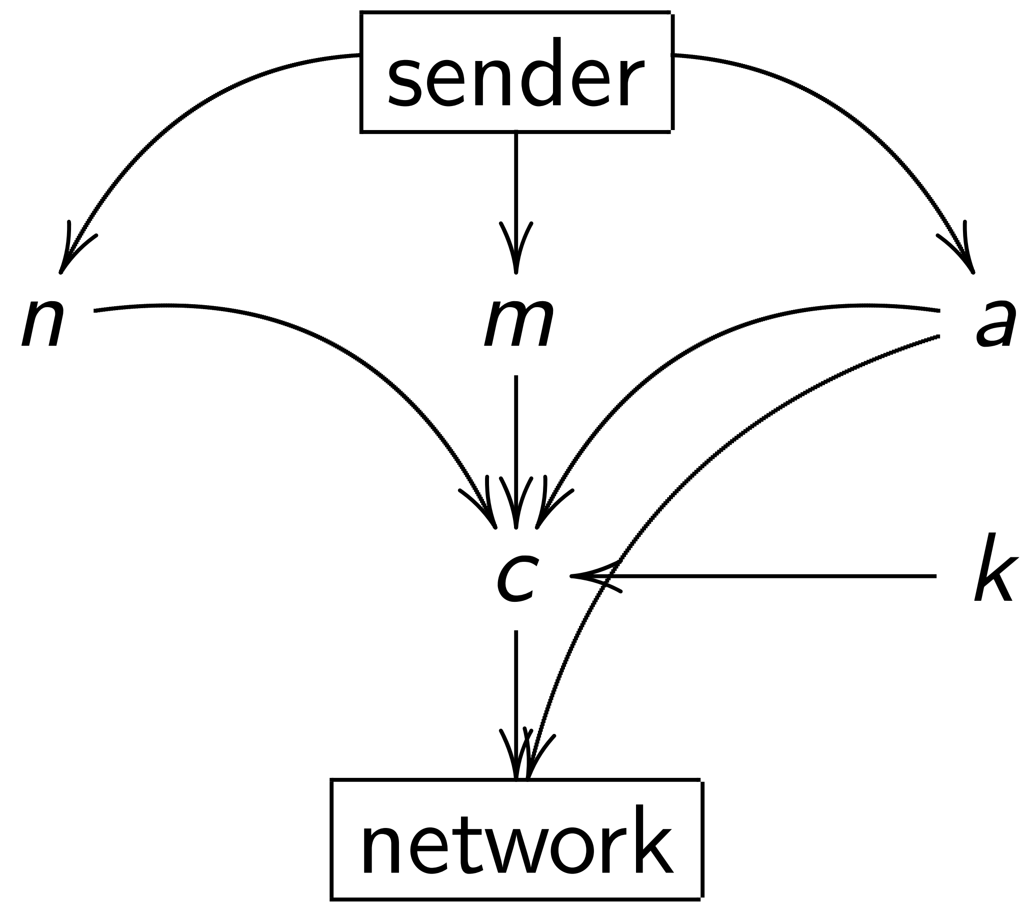
a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

No problem repeating a.

Secret message numbers



k: secret key.

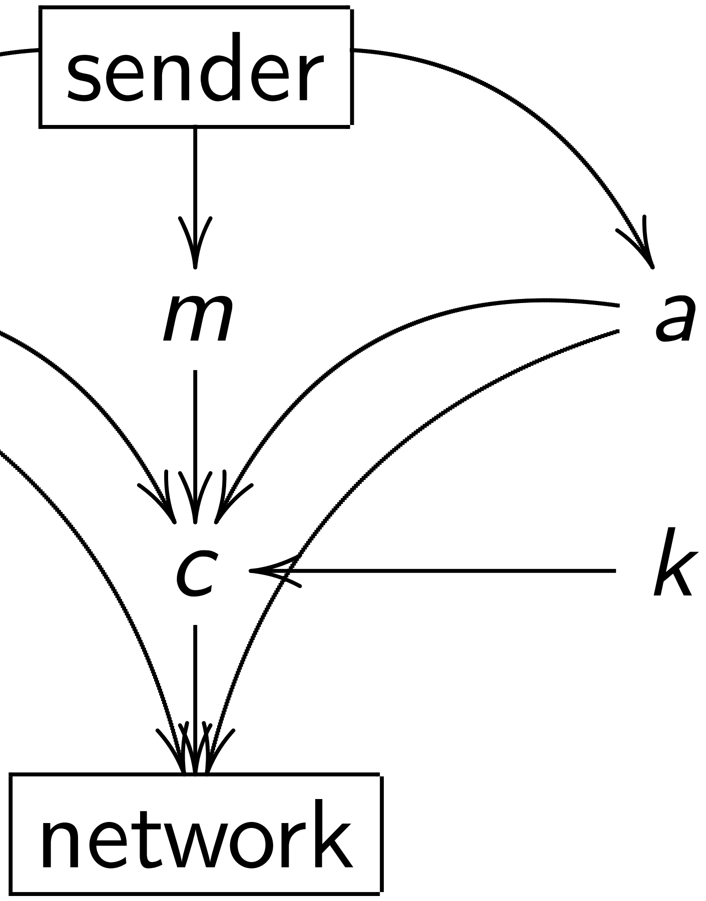
n: secret message number.

a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

ed data



t key.

c message number.

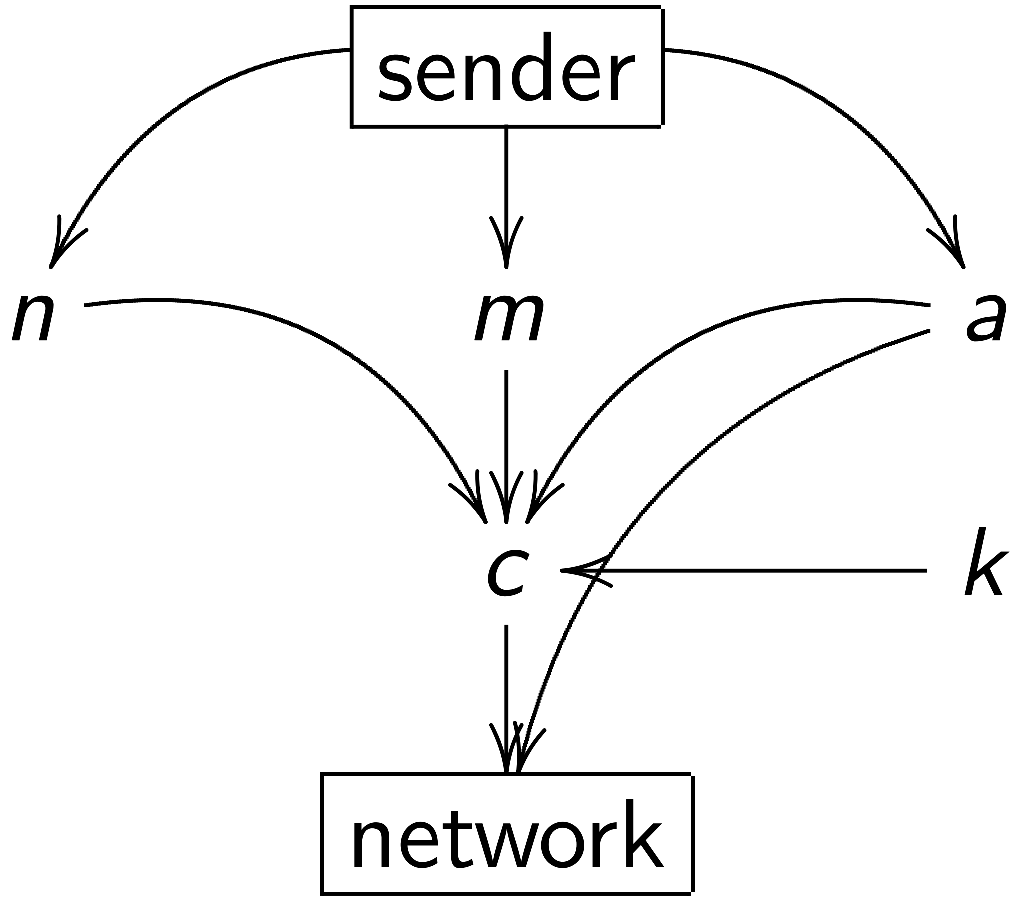
ble-length associated data.

able-length plaintext.

ble-length ciphertext.

lem repeating *a*.

Secret message numbers



k: secret key.

n: secret message number.

a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

What is

Plaintex

associat

messag

Forge (n

receiver

legitimat

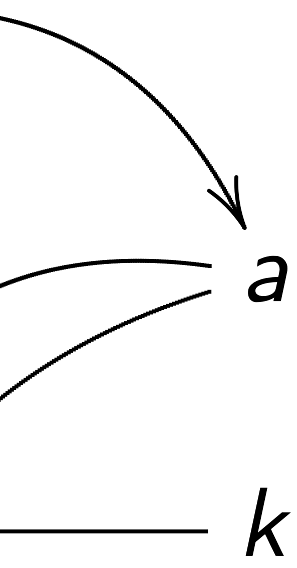
“INT-PT

(integrit

protectio

Stronge

Forge at



number.

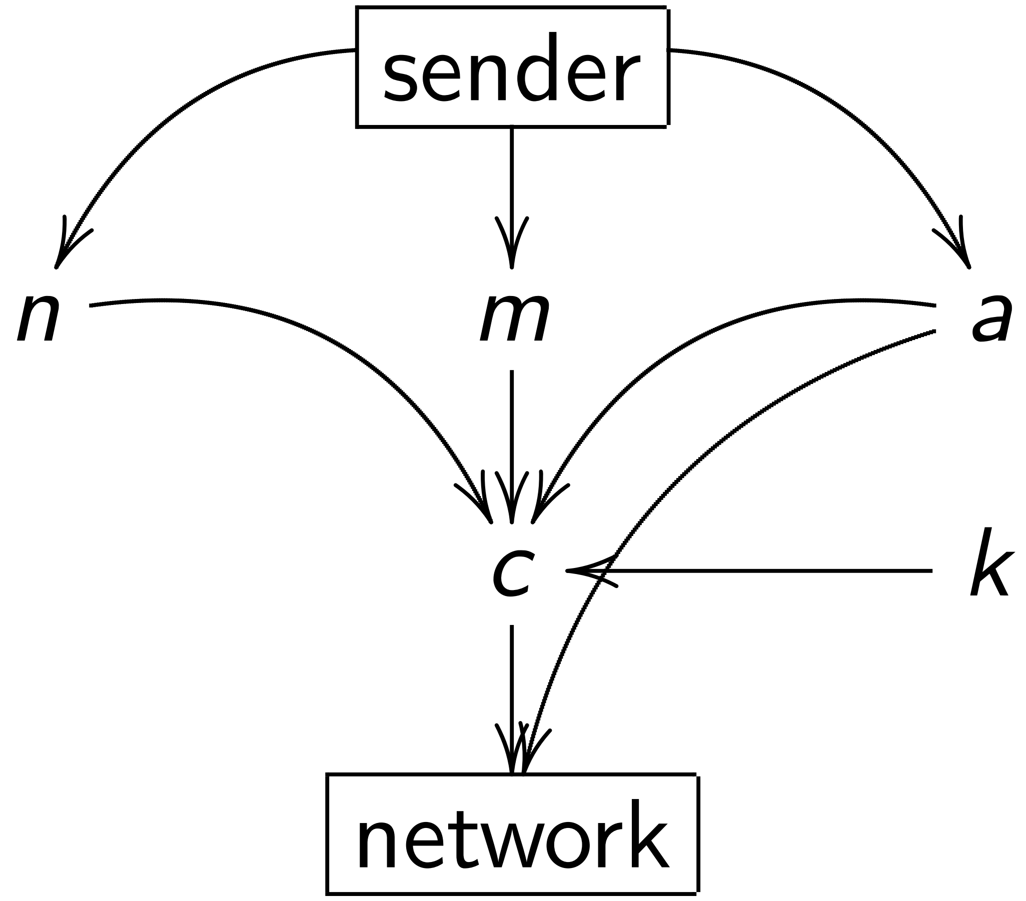
associated data.

plaintext.

ciphertext.

ting *a*.

Secret message numbers



k: secret key.

n: secret message number.

a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

What is the attack

Plaintext corrupt

associated-data c

message-number

Forge (n, m, a) tha

receiver accepts bu

legitimate sender n

"INT-PTXT"

(integrity of plaint

protection against

Stronger goal:

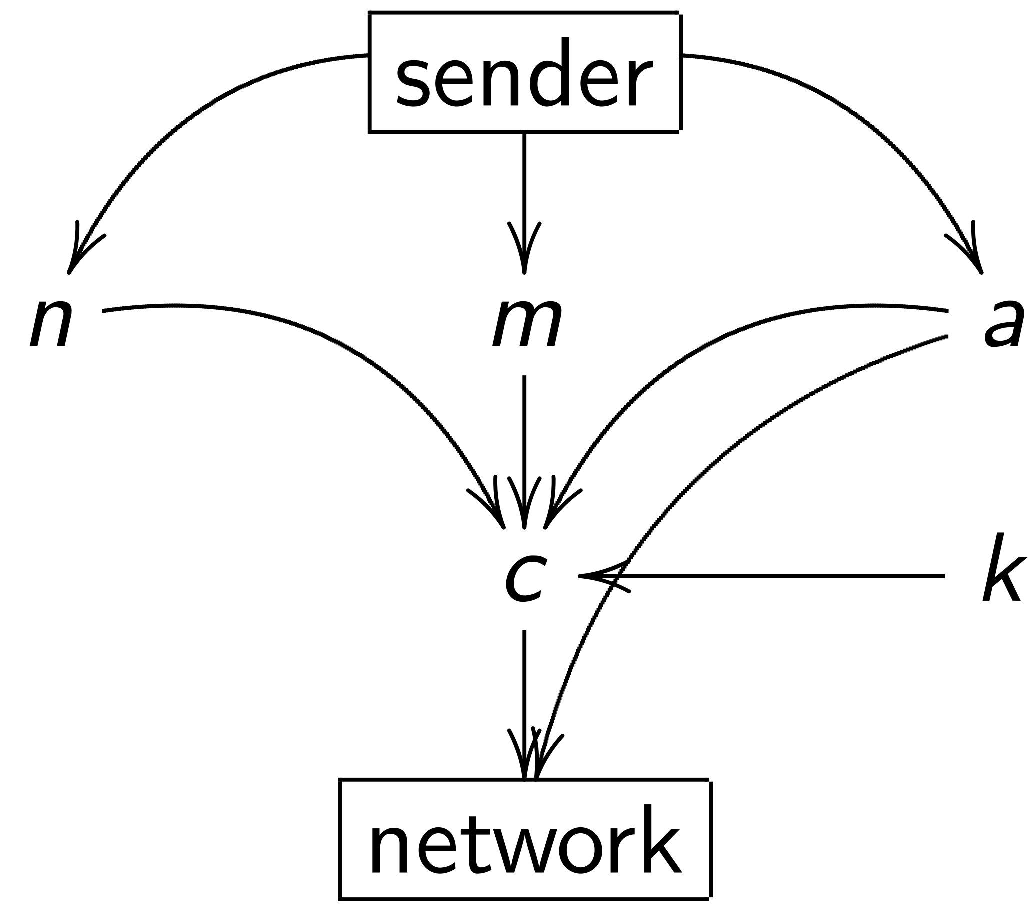
Forge at least f m

d data.

.

t.

Secret message numbers



k: secret key.

n: secret message number.

a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

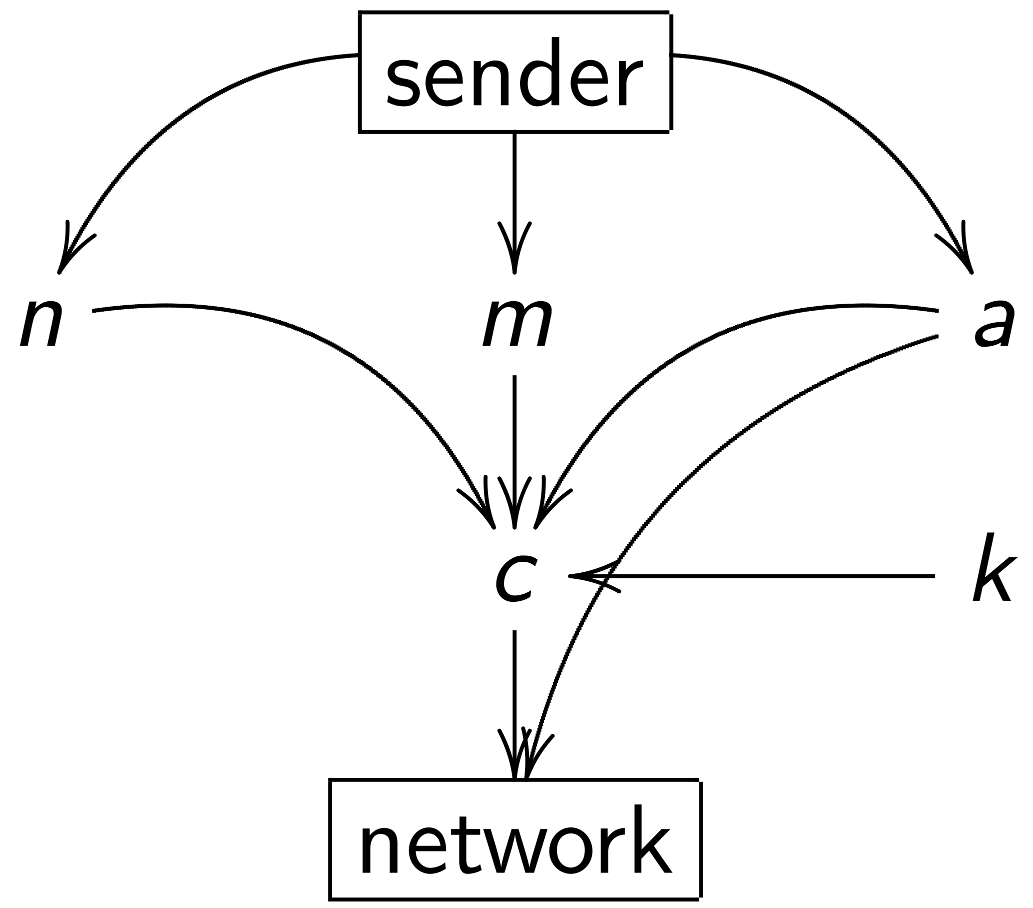
What is the attacker's goal?

Plaintext corruption,
associated-data corruptio
message-number corruptio
Forge (n, m, a) that
receiver accepts but that
legitimate sender never encr

"INT-PTXT"
(integrity of plaintexts) mea
protection against such atta

Stronger goal:
Forge at least f messages.

Secret message numbers



k: secret key.

n: secret message number.

a: variable-length associated data.

m: variable-length plaintext.

c: variable-length ciphertext.

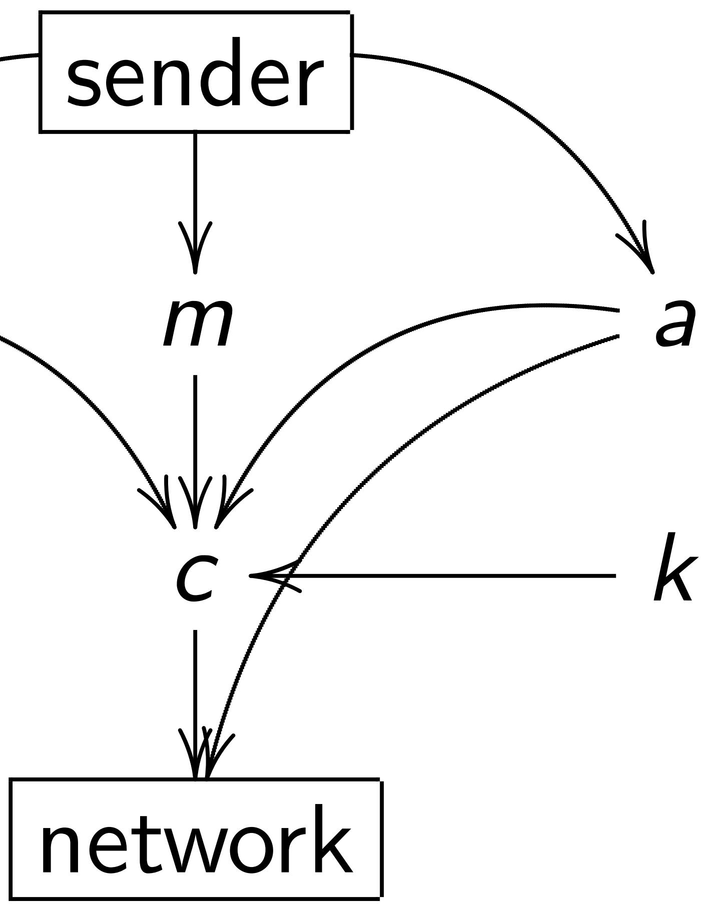
What is the attacker's goal?

Plaintext corruption, associated-data corruption, message-number corruption. Forge (n, m, a) that receiver accepts but that legitimate sender never encrypted.

"INT-PTXT" (integrity of plaintexts) means protection against such attacks.

Stronger goal:
Forge at least f messages.

nessage numbers



t key.

t message number.

ble-length associated data.

able-length plaintext.

ble-length ciphertext.

What is the attacker's goal?

Plaintext corruption,
associated-data corruption,
message-number corruption.
Forge (n, m, a) that
receiver accepts but that
legitimate sender never encrypted.

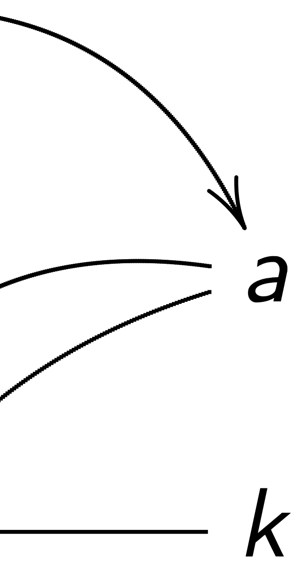
"INT-PTXT"
(integrity of plaintexts) means
protection against such attacks.

Stronger goal:
Forge at least f messages.

Ciphert
Forge c
receiver
legitimat

"INT-CT
(integrit
protectio

umbers



number.

associated data.

plaintext.

ciphertext.

What is the attacker's goal?

Plaintext corruption, associated-data corruption, message-number corruption.
Forge (n, m, a) that
receiver accepts but that
legitimate sender never encrypted.

"INT-PTXT"
(integrity of plaintexts) means
protection against such attacks.

Stronger goal:
Forge at least f messages.

Ciphertext corru
Forge c that
receiver accepts bu
legitimate sender

"INT-CTXT"
(integrity of ciphe
protection against

d data.

T
t.

What is the attacker's goal?

Plaintext corruption, associated-data corruption, message-number corruption.
Forge (n, m, a) that
receiver accepts but that
legitimate sender never encrypted.

"INT-PTXT"
(integrity of plaintexts) means
protection against such attacks.

Stronger goal:
Forge at least f messages.

Ciphertext corruption.
Forge c that
receiver accepts but that
legitimate sender never prod

"INT-CTXT"
(integrity of ciphertexts) me
protection against such atta

What is the attacker's goal?

Plaintext corruption, associated-data corruption, message-number corruption. Forge (n, m, a) that receiver accepts but that legitimate sender never encrypted.

"INT-PTXT" (integrity of plaintexts) means protection against such attacks.

Stronger goal: Forge at least f messages.

Ciphertext corruption. Forge c that receiver accepts but that legitimate sender never produced.

"INT-CTXT" (integrity of ciphertexts) means protection against such attacks.

What is the attacker's goal?

Plaintext corruption,
associated-data corruption,
message-number corruption.
Forge (n, m, a) that
receiver accepts but that
legitimate sender never encrypted.

"INT-PTXT"
(integrity of plaintexts) means
protection against such attacks.

Stronger goal:
Forge at least f messages.

Ciphertext corruption.
Forge c that
receiver accepts but that
legitimate sender never produced.

"INT-CTXT"
(integrity of ciphertexts) means
protection against such attacks.

Ciphertext prediction.
Distinguish c from
uniform random string.

What is the attacker's goal?

Plaintext corruption, associated-data corruption, message-number corruption. Forge (n, m, a) that receiver accepts but that legitimate sender never encrypted.

"INT-PTXT" (integrity of plaintexts) means protection against such attacks.

Stronger goal: Forge at least f messages.

Ciphertext corruption. Forge c that receiver accepts but that legitimate sender never produced.

"INT-CTXT" (integrity of ciphertexts) means protection against such attacks.

Ciphertext prediction. Distinguish c from uniform random string.

Is it better to *randomly pad* or *zero-pad* a strong 112-bit MAC to 128 bits?

the attacker's goal?

xt corruption,
ted-data corruption,
e-number corruption.

n, m, a) that
accepts but that
te sender never encrypted.

TXT"
y of plaintexts) means
on against such attacks.

r goal:
least f messages.

Ciphertext corruption.
Forge c that
receiver accepts but that
legitimate sender never produced.

"INT-CTXT"
(integrity of ciphertexts) means
protection against such attacks.

Ciphertext prediction.
Distinguish c from
uniform random string.

Is it better to
randomly pad or *zero-pad* a
strong 112-bit MAC to 128 bits?

Replay.
Convinc
legitima
than leg

ker's goal?

tion,

corruption,

r corruption.

at

ut that

never encrypted.

xexts) means

such attacks.

nessages.

Ciphertext corruption.

Forge c that

receiver accepts but that

legitimate sender never produced.

"INT-CTXT"

(integrity of ciphertexts) means

protection against such attacks.

Ciphertext prediction.

Distinguish c from

uniform random string.

Is it better to

randomly pad or *zero-pad* a

strong 112-bit MAC to 128 bits?

Replay.

Convince receiver

legitimate $(n, m, a$

than legitimate se

n,
on.

ypted.

ns
cks.

Ciphertext corruption.
Forge c that
receiver accepts but that
legitimate sender never produced.

"INT-CTXT"
(integrity of ciphertexts) means
protection against such attacks.

Ciphertext prediction.
Distinguish c from
uniform random string.

Is it better to
randomly pad or *zero-pad* a
strong 112-bit MAC to 128 bits?

Replay.
Convince receiver to accept
legitimate (n, m, a) more tim
than legitimate sender sent

Ciphertext corruption.

Forge c that receiver accepts but that legitimate sender never produced.

"INT-CTXT" (integrity of ciphertexts) means protection against such attacks.

Ciphertext prediction.

Distinguish c from uniform random string.

Is it better to *randomly pad* or *zero-pad* a strong 112-bit MAC to 128 bits?

Replay.

Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Ciphertext corruption.
Forge c that receiver accepts but that legitimate sender never produced.

"INT-CTXT" (integrity of ciphertexts) means protection against such attacks.

Ciphertext prediction.
Distinguish c from uniform random string.

Is it better to *randomly pad* or *zero-pad* a strong 112-bit MAC to 128 bits?

Replay.
Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.
Convince receiver to accept legitimate messages out of order.

Ciphertext corruption.
Forge c that receiver accepts but that legitimate sender never produced.

"INT-CTXT" (integrity of ciphertexts) means protection against such attacks.

Ciphertext prediction.
Distinguish c from uniform random string.

Is it better to *randomly pad* or *zero-pad* a strong 112-bit MAC to 128 bits?

Replay.
Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.
Convince receiver to accept legitimate messages out of order.

Sabotage.
Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Ciphertext corruption.
Forge c that receiver accepts but that legitimate sender never produced.

"INT-CTXT" (integrity of ciphertexts) means protection against such attacks.

Ciphertext prediction.
Distinguish c from uniform random string.

Is it better to *randomly pad* or *zero-pad* a strong 112-bit MAC to 128 bits?

Replay.
Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.
Convince receiver to accept legitimate messages out of order.

Sabotage.
Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

ext corruption.

that
accepts but that
te sender never produced.

TXT”
y of ciphertexts) means
on against such attacks.

ext prediction.

ish c from
random string.

ter to
y pad or *zero-pad* a
12-bit MAC to 128 bits?

Replay.

Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.

Convince receiver to accept legitimate messages out of order.

Sabotage.

Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintex

Figure o

ption.

ut that

never produced.

rtexts) means

such attacks.

ction.

n

tring.

zero-pad a

AC to 128 bits?

Replay.

Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.

Convince receiver to accept legitimate messages out of order.

Sabotage.

Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintext espiona

Figure out user's s

luced.

ans

cks.

bits?

Replay.

Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.

Convince receiver to accept legitimate messages out of order.

Sabotage.

Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintext espionage.

Figure out user's secret mes

Replay.

Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.

Convince receiver to accept legitimate messages out of order.

Sabotage.

Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintext espionage.

Figure out user's secret message.

Replay.
Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.
Convince receiver to accept legitimate messages out of order.

Sabotage.
Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret message number.

Replay.
Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.
Convince receiver to accept legitimate messages out of order.

Sabotage.
Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret message number.

Traditional crypto view:
It's okay to use a counter as a message number.
Count is public anyway.

Replay.
Convince receiver to accept legitimate (n, m, a) more times than legitimate sender sent it.

Reordering.
Convince receiver to accept legitimate messages out of order.

Sabotage.
Prevent receiver from seeing (n, m, a) as often as sender sent it: flood radio, switch, CPU, etc.

Typically delegate solutions to higher-level protocols, but is this optimal?

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret message number.

Traditional crypto view:
It's okay to use a counter as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better, but how much does it leak?
Should encrypt by default.

e receiver to accept
te (n, m, a) more times
itimate sender sent it.

ring.
e receiver to accept
te messages out of order.

ge.
receiver from seeing
as often as sender sent
radio, switch, CPU, etc.

y delegate solutions
r-level protocols,
his optimal?

Plaintext espionage.
Figure out user’s secret message.

Message-number espionage.
Figure out user’s secret
message number.

Traditional crypto view:
It’s okay to use a counter
as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better,
but how much does it leak?
Should encrypt by default.

What ar

Extensi
Are 80-b
Are 128-

to accept
) more times
nder sent it.

to accept
es out of order.

rom seeing
as sender sent
itch, CPU, etc.

solutions
tocols,
?

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret
message number.

Traditional crypto view:
It's okay to use a counter
as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better,
but how much does it leak?
Should encrypt by default.

What are the atta

Extensive compu
Are 80-bit keys ad
Are 128-bit keys a

nes
it.

order.

;
sent
, etc.

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret message number.

Traditional crypto view:
It's okay to use a counter as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better, but how much does it leak?
Should encrypt by default.

What are the attacker's reso

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Plaintext espionage.

Figure out user's secret message.

Message-number espionage.

Figure out user's secret message number.

Traditional crypto view:
It's okay to use a counter as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better, but how much does it leak?
Should encrypt by default.

What are the attacker's resources?

Extensive computation.

Are 80-bit keys adequate?
Are 128-bit keys adequate?

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret message number.

Traditional crypto view:
It's okay to use a counter as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better, but how much does it leak?
Should encrypt by default.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

Plaintext espionage.
Figure out user's secret message.

Message-number espionage.
Figure out user's secret message number.

Traditional crypto view:
It's okay to use a counter as a message number.
Count is public anyway.

Counterarguments:
Did attacker see everything?
Maybe timestamp is better, but how much does it leak?
Should encrypt by default.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

xt espionage.

ut user's secret message.

e-number espionage.

ut user's secret

number.

nal crypto view:

y to use a counter

ssage number.

s public anyway.

arguments:

cker see everything?

imestamp is better,

much does it leak?

encrypt by default.

What are the attacker's resources?

Extensive computation.

Are 80-bit keys adequate?

Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

Main co

1. Large

User doe

better p

age.

secret message.

r **espionage.**

secret

view:

counter

ber.

yway.

:

verything?

is better,

es it leak?

default.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

Main counterargu

1. Larger keys are
User doesn't actua
better performanc

sage.

ge.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

Main counterarguments:

1. Larger keys are cheap en
User doesn't actually need
better performance.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

What are the attacker's resources?

Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?

Main arguments for small keys:

1. Smaller keys are cheaper.

2. Attack cost outweighs economic benefit of breaking key, so no sensible attacker will carry out a 2^{80} attack.

Maybe 64-bit keys are enough.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

3. Some attackers carry out attacks that are feasible but not economically rational.

What attacks are feasible?

e the attacker's resources?

ve computation.

bit keys adequate?

-bit keys adequate?

guments for small keys:

ler keys are cheaper.

ck cost outweighs
c benefit of breaking key,
nsible attacker will
t a 2^{80} attack.

64-bit keys are enough.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

3. Some attackers carry out attacks that are feasible but not economically rational.

What attacks are feasible?

Back-of-

2^{57} watt
atmosph

2^{44} watt

2^{26} watt
costing 2

1 watt:
2^{68} bit c
using ma

cker's resources?

tation.

equate?

dequate?

or small keys:

e cheaper.

tweighs

of breaking key,

acker will

tack.

s are enough.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

3. Some attackers carry out attacks that are feasible but not economically rational.

What attacks are feasible?

Back-of-the-envelo

2^{57} watts: receive

atmosphere from t

2^{44} watts: world p

2^{26} watts: one co

costing 2^{30} dollars

1 watt: power for

2^{68} bit operations

using mass-market

ources?

eys:

.

g key,

gh.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

3. Some attackers carry out attacks that are feasible but not economically rational.

What attacks are feasible?

Back-of-the-envelope figures

2^{57} watts: received by Earth atmosphere from the Sun.

2^{44} watts: world power usag

2^{26} watts: one computer cen costing 2^{30} dollars.

1 watt: power for 2^{68} bit operations per year using mass-market GPUs.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

3. Some attackers carry out attacks that are feasible but not economically rational.

What attacks are feasible?

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for 2^{68} bit operations per year using mass-market GPUs.

Main counterarguments:

1. Larger keys are cheap enough. User doesn't actually need better performance.

2. Attacker's cost-benefit ratio is improved by multiple-user attacks, multiple forgeries, etc.

3. Some attackers carry out attacks that are feasible but not economically rational.

What attacks are feasible?

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for 2^{68} bit operations per year using mass-market GPUs.

Scalable quantum computers. 2^{64} simple quantum operations to find a 128-bit key using Grover's algorithm.

unterarguments:

er keys are cheap enough.
esn't actually need
erformance.

cker's cost-benefit ratio
ved by
-user attacks,
forgeries, etc.

e attackers carry out
that are feasible
economically rational.

tacks are feasible?

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for
2^{68} bit operations per year
using mass-market GPUs.

Scalable quantum computers.
2^{64} simple quantum operations
to find a 128-bit key
using Grover's algorithm.

Many m

Some de
"switch
Other de
eliminati
adds rob

ments:

cheap enough.
ally need
e.

-benefit ratio

cks,
etc.

carry out
easible
ally rational.

feasible?

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for
2^{68} bit operations per year
using mass-market GPUs.

Scalable quantum computers.
2^{64} simple quantum operations
to find a 128-bit key
using Grover's algorithm.

Many messages.

Some designers bl
"switch keys after
Other designers ar
eliminating such re
adds robustness.

ough.

atio

al.

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for
2^{68} bit operations per year
using mass-market GPUs.

Scalable quantum computers.
2^{64} simple quantum operations
to find a 128-bit key
using Grover's algorithm.

Many messages.
Some designers blame the u
"switch keys after 2^{20} messa
Other designers argue that
eliminating such requiremen
adds robustness.

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for
2^{68} bit operations per year
using mass-market GPUs.

Scalable quantum computers.
2^{64} simple quantum operations
to find a 128-bit key
using Grover's algorithm.

Many messages.
Some designers blame the user:
"switch keys after 2^{20} messages".
Other designers argue that
eliminating such requirements
adds robustness.

Back-of-the-envelope figures:

2^{57} watts: received by Earth's atmosphere from the Sun.

2^{44} watts: world power usage.

2^{26} watts: one computer center costing 2^{30} dollars.

1 watt: power for 2^{68} bit operations per year using mass-market GPUs.

Scalable quantum computers.
2^{64} simple quantum operations to find a 128-bit key using Grover's algorithm.

Many messages.
Some designers blame the user: "switch keys after 2^{20} messages". Other designers argue that eliminating such requirements adds robustness.

Chosen plaintexts,
chosen ciphertexts,
chosen message numbers.
Consensus:
Unacceptable to blame the user.
All ciphers must be safe against chosen-plaintext attacks and against chosen-ciphertext attacks.

-the-envelope figures:

ts: received by Earth's
here from the Sun.

ts: world power usage.

ts: one computer center
2^{30} dollars.

power for
operations per year
ass-market GPUs.

e quantum computers.
ble quantum operations
n 128-bit key
rover's algorithm.

Many messages.
Some designers blame the user:
"switch keys after 2^{20} messages".
Other designers argue that
eliminating such requirements
adds robustness.

Chosen plaintexts,
chosen ciphertexts,
chosen message numbers.
Consensus:
Unacceptable to blame the user.
All ciphers must be safe against
chosen-plaintext attacks and
against chosen-ciphertext attacks.

Many u
degrade

ope figures:

d by Earth's
the Sun.

power usage.

mputer center
s.

per year
t GPUs.

m computers.
m operations
key
orithm.

Many messages.
Some designers blame the user:
"switch keys after 2^{20} messages".
Other designers argue that
eliminating such requirements
adds robustness.

Chosen plaintexts,
chosen ciphertexts,
chosen message numbers.
Consensus:
Unacceptable to blame the user.
All ciphers must be safe against
chosen-plaintext attacks and
against chosen-ciphertext attacks.

Many users. How
degrade with num

s:

n's

ge.

nter

ters.

ons

Many messages.

Some designers blame the user: "switch keys after 2^{20} messages".

Other designers argue that eliminating such requirements adds robustness.

Chosen plaintexts, chosen ciphertexts, chosen message numbers.

Consensus:

Unacceptable to blame the user.

All ciphers must be safe against chosen-plaintext attacks and against chosen-ciphertext attacks.

Many users. How does sec degrade with number of key

Many messages.
Some designers blame the user: "switch keys after 2^{20} messages". Other designers argue that eliminating such requirements adds robustness.

Chosen plaintexts,
chosen ciphertexts,
chosen message numbers.
Consensus:
Unacceptable to blame the user. All ciphers must be safe against chosen-plaintext attacks and against chosen-ciphertext attacks.

Many users. How does security degrade with number of keys?

Many messages.
Some designers blame the user: "switch keys after 2^{20} messages". Other designers argue that eliminating such requirements adds robustness.

Chosen plaintexts, chosen ciphertexts, chosen message numbers.
Consensus:
Unacceptable to blame the user. All ciphers must be safe against chosen-plaintext attacks and against chosen-ciphertext attacks.

Many users. How does security degrade with number of keys?

Repeated message numbers.
Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

essages.

esigners blame the user:

keys after 2^{20} messages".

esigners argue that

ing such requirements

oustness.

plaintexts,

ciphertexts,

message numbers.

us:

otable to blame the user.

ers must be safe against

plaintext attacks and

chosen-ciphertext attacks.

Many users. How does security degrade with number of keys?

Repeated message numbers. Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Softwar

Typical

secret br

secret m

Also, on

secret m

ame the user:
2^{20} messages".
rgue that
equirements

s,
ts,
numbers.

lame the user.
e safe against
ttacks and
phertext attacks.

Many users. How does security degrade with number of keys?

Repeated message numbers. Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Software side ch
Typical culprits:
secret branches,
secret memory ad
Also, on some CP
secret multiplicati

ser:

ages”.

ts

user.

inst

d

tacks.

Many users. How does security degrade with number of keys?

Repeated message numbers. Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Software side channels. Typical culprits: secret branches, secret memory addresses. Also, on some CPUs, secret multiplication inputs.

Many users. How does security degrade with number of keys?

Repeated message numbers. Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Software side channels. Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Many users. How does security degrade with number of keys?

Repeated message numbers. Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Software side channels. Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels. Power consumption, electromagnetic radiation, etc.

Many users. How does security degrade with number of keys?

Repeated message numbers.
Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Software side channels.
Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels.
Power consumption,
electromagnetic radiation, etc.

Faults. Flip bits in computation.

Many users. How does security degrade with number of keys?

Repeated message numbers.
Minimum impact: Attacker sees whether (n, m, a) is repeated.

Examples of larger impact for many ciphers:

Leak number of shared initial blocks of plaintext.

Leak xor of first non-shared block.

Allow forgery under this n.

Allow forgery under any n'.

Software side channels.
Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels.
Power consumption,
electromagnetic radiation, etc.

Faults. Flip bits in computation.

Thefts and monitors.
Attacker steals secret keys.
Can we still protect *past* communication?

sers. How does security
with number of keys?

ed message numbers.
m impact: Attacker sees
(n, m, a) is repeated.

es of larger impact
y ciphers:

mber of shared initial
f plaintext.

r of first non-shared block.

rgery under this n.

rgery under any n'.

Software side channels.
Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels.
Power consumption,
electromagnetic radiation, etc.

Faults. Flip bits in computation.

Thefts and monitors.
Attacker steals secret keys.
Can we still protect
past communication?

<u>What pe</u>

Typical p
for ASIC

Low ene

Low pow

Low area
loosely p
“gate eq

High thr
(bytes p

Low late
very loos

v does security

ber of keys?

ge numbers.

Attacker sees

is repeated.

r impact

hared initial

t.

on-shared block.

er this n.

er any n'.

Software side channels.
Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels.
Power consumption,
electromagnetic radiation, etc.

Faults. Flip bits in computation.

Thefts and monitors.
Attacker steals secret keys.
Can we still protect
past communication?

What performance

Typical performan
for ASICs:

Low energy (joule

Low power (watts

Low area (square
loosely predicted b
"gate equivalents"

High throughput
(bytes per second

Low latency (seco
very loosely predi

urity
s?

ers.

sees
d.

al

block.

Software side channels.
Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels.
Power consumption,
electromagnetic radiation, etc.

Faults. Flip bits in computation.

Thefts and monitors.
Attacker steals secret keys.
Can we still protect
past communication?

<u>What performance is measu</u>

Typical performance metrics
for ASICs:

Low energy (joules) per byte

Low power (watts).

Low area (square micromete
loosely predicted by
“gate equivalents”).

High throughput
(bytes per second).

Low latency (seconds;
very loosely predicted by cyc

Software side channels.
Typical culprits:
secret branches,
secret memory addresses.
Also, on some CPUs,
secret multiplication inputs.

Hardware side channels.
Power consumption,
electromagnetic radiation, etc.

Faults. Flip bits in computation.

Thefts and monitors.
Attacker steals secret keys.
Can we still protect
past communication?

What performance is measured?

Typical performance metrics
for ASICs:

Low energy (joules) per byte.

Low power (watts).

Low area (square micrometers;
loosely predicted by
"gate equivalents").

High throughput
(bytes per second).

Low latency (seconds;
very loosely predicted by cycles).

e side channels.

culprits:

ranches,

nemory addresses.

some CPUs,

nultiplication inputs.

re side channels.

onsumption,

nagnetic radiation, etc.

Flip bits in computation.

and monitors.

steals secret keys.

still protect

mmunication?

What performance is measured?

Typical performance metrics for ASICs:

Low energy (joules) per byte.

Low power (watts).

Low area (square micrometers; loosely predicted by “gate equivalents”).

High throughput (bytes per second).

Low latency (seconds; very loosely predicted by cycles).

Similar r

FPGAs a

For ASIC

through

is not a

without

Paralleliz

or across

for arbit

annels.

dresses.

Us,

on inputs.

hannels.

n,

adiation, etc.

n computation.

tors.

cret keys.

ct

on?

What performance is measured?

Typical performance metrics for ASICs:

Low energy (joules) per byte.

Low power (watts).

Low area (square micrometers; loosely predicted by "gate equivalents").

High throughput (bytes per second).

Low latency (seconds; very loosely predicted by cycles).

Similar metrics for FPGAs and softwa

For ASICs and FP throughput per se is not a useful met without limit on a

Parallelize across b or across independ for arbitrarily high tc.

ation.

What performance is measured?

Typical performance metrics for ASICs:

Low energy (joules) per byte.

Low power (watts).

Low area (square micrometers; loosely predicted by “gate equivalents”).

High throughput (bytes per second).

Low latency (seconds; very loosely predicted by cycles).

Similar metrics for FPGAs and software.

For ASICs and FPGAs, throughput per se is not a useful metric without limit on area (or po

Parallelize across blocks or across independent messa for arbitrarily high throughp

What performance is measured?

Typical performance metrics
for ASICs:

Low energy (joules) per byte.

Low power (watts).

Low area (square micrometers;
loosely predicted by
"gate equivalents").

High throughput
(bytes per second).

Low latency (seconds;
very loosely predicted by cycles).

Similar metrics for
FPGAs and software.

For ASICs and FPGAs,
throughput per se
is not a useful metric
without limit on area (or power).

Parallelize across blocks
or across independent messages
for arbitrarily high throughput.

What performance is measured?

Typical performance metrics
for ASICs:

Low energy (joules) per byte.

Low power (watts).

Low area (square micrometers;
loosely predicted by
“gate equivalents”).

High throughput
(bytes per second).

Low latency (seconds;
very loosely predicted by cycles).

Similar metrics for
FPGAs and software.

For ASICs and FPGAs,
throughput per se
is not a useful metric
without limit on area (or power).

Parallelize across blocks
or across independent messages
for arbitrarily high throughput.

Fix: measure
ratio of area and throughput, i.e.,
product of area and time per byte.

erformance is measured?

performance metrics

Cs:

rgy (joules) per byte.

ver (watts).

a (square micrometers;

predicted by

quivalents").

roughput

er second).

ency (seconds;

sely predicted by cycles).

Similar metrics for FPGAs and software.

For ASICs and FPGAs, throughput per se is not a useful metric without limit on area (or power).

Parallelize across blocks or across independent messages for arbitrarily high throughput.

Fix: measure ratio of area and throughput, i.e., product of area and time per byte.

What op

Authent

encrypt

Cost per

far below

Send va

data, or

"Encryp

cost of c

e is measured?

ce metrics

s) per byte.

).

micrometers;

oy

).

).

nds;

cted by cycles).

Similar metrics for FPGAs and software.

For ASICs and FPGAs, throughput per se is not a useful metric without limit on area (or power).

Parallelize across blocks or across independent messages for arbitrarily high throughput.

Fix: measure ratio of area and throughput, i.e., product of area and time per byte.

What operations a

Authenticate onl

encrypt and auth

Cost per byte of *a*

far below cost per

Send valid data,

data, or receive i

“Encrypt then MA

cost of decryption

red?

s

e.

ers;

cles).

Similar metrics for FPGAs and software.

For ASICs and FPGAs, throughput per se is not a useful metric without limit on area (or power).

Parallelize across blocks or across independent messages for arbitrarily high throughput.

Fix: measure ratio of area and throughput, i.e., product of area and time per byte.

What operations are measur

Authenticate only, or encrypt and authenticate?
Cost per byte of *a* can be far below cost per byte of *m*

Send valid data, receive va data, or receive invalid da
"Encrypt then MAC" skips cost of decryption for forger

Similar metrics for FPGAs and software.

For ASICs and FPGAs, throughput per se is not a useful metric without limit on area (or power).

Parallelize across blocks or across independent messages for arbitrarily high throughput.

Fix: measure ratio of area and throughput, i.e., product of area and time per byte.

What operations are measured?

Authenticate only, or encrypt and authenticate? Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data? "Encrypt then MAC" skips cost of decryption for forgeries.

Similar metrics for FPGAs and software.

For ASICs and FPGAs, throughput per se is not a useful metric without limit on area (or power).

Parallelize across blocks or across independent messages for arbitrarily high throughput.

Fix: measure ratio of area and throughput, i.e., product of area and time per byte.

What operations are measured?

Authenticate only, or encrypt and authenticate? Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data? "Encrypt then MAC" skips cost of decryption for forgeries.

Different area targets: encryption/authentication circuit; verification/decryption circuit; circuit that can dynamically select either operation.

metrics for
and software.

Cs and FPGAs,
out per se
useful metric
limit on area (or power).

ze across blocks
s independent messages
rarily high throughput.

asure
area and throughput, i.e.,
of area and time per byte.

What operations are measured?

Authenticate only, or encrypt and authenticate?
Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data?
"Encrypt then MAC" skips cost of decryption for forgeries.

Different area targets: encryption/authentication circuit; verification/decryption circuit; circuit that can dynamically select either operation.

Plaintex
and ass
Huge im

Warning
"overhea
Cipher w
can be c

r

are.

GAs,

tric

rea (or power).

blocks

dent messages

throughput.

throughput, i.e.,

nd time per byte.

What operations are measured?

Authenticate only, or encrypt and authenticate?
Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data?
"Encrypt then MAC" skips cost of decryption for forgeries.

Different area targets: encryption/authentication circuit; verification/decryption circuit; circuit that can dynamically select either operation.

Plaintext length

and associated-d

Huge impact on p

Warning: Do not

"overhead" of two

Cipher with larger

can be consistentl

wer).

nges
ut.

t, i.e.,
r byte.

What operations are measured?

Authenticate only, or encrypt and authenticate?
Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data?
"Encrypt then MAC" skips cost of decryption for forgeries.

Different area targets:
encryption/authentication circuit;
verification/decryption circuit;
circuit that can dynamically select either operation.

Plaintext length
and associated-data lengt
Huge impact on performanc

Warning: Do not solely com
"overhead" of two ciphers.
Cipher with larger "overhea
can be consistently faster.

What operations are measured?

Authenticate only, or encrypt and authenticate? Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data? "Encrypt then MAC" skips cost of decryption for forgeries.

Different area targets: encryption/authentication circuit; verification/decryption circuit; circuit that can dynamically select either operation.

Plaintext length and associated-data length. Huge impact on performance.

Warning: Do not solely compare "overhead" of two ciphers. Cipher with larger "overhead" can be consistently faster.

What operations are measured?

Authenticate only, or encrypt and authenticate?
Cost per byte of *a* can be far below cost per byte of *m*.

Send valid data, receive valid data, or receive invalid data?
"Encrypt then MAC" skips cost of decryption for forgeries.

Different area targets:
encryption/authentication circuit;
verification/decryption circuit;
circuit that can dynamically select either operation.

Plaintext length and associated-data length.
Huge impact on performance.

Warning: Do not solely compare "overhead" of two ciphers. Cipher with larger "overhead" can be consistently faster.

Number of inputs.
e.g. reduce latency by processing several AES-CBC messages in parallel. Simplest if many messages have the same length.

perations are measured?

ticate only, or
and authenticate?

r byte of *a* can be
w cost per byte of *m*.

alid data, receive valid
r receive invalid data?

t then MAC" skips
decryption for forgeries.

t area targets:
on/authentication circuit;
ion/decryption circuit;
hat can dynamically select
peration.

Plaintext length
and associated-data length.
Huge impact on performance.

Warning: Do not solely compare "overhead" of two ciphers. Cipher with larger "overhead" can be consistently faster.

Number of inputs.
e.g. reduce latency by processing several AES-CBC messages in parallel. Simplest if many messages have the same length.

Numbe
Most (n
expect p
"expand

Warning
"agility"
Cipher w
can be c

are measured?

y, or
henticate?
can be
byte of *m*.

receive valid
nvalid data?
AC" skips
for forgeries.

gets:
tication circuit;
ption circuit;
ynamically select

Plaintext length
and associated-data length.
Huge impact on performance.

Warning: Do not solely compare "overhead" of two ciphers. Cipher with larger "overhead" can be consistently faster.

Number of inputs.
e.g. reduce latency by processing several AES-CBC messages in parallel. Simplest if many messages have the same length.

Number of times
Most (not all) cip
expect precomput
"expanded keys".

Warning: Do not
"agility" of two ci
Cipher with better
can be consistentl

red?

?

n.

alid
ta?

ies.

ircuit;
it;
select

Plaintext length
and associated-data length.
Huge impact on performance.

Warning: Do not solely compare
“overhead” of two ciphers.
Cipher with larger “overhead”
can be consistently faster.

Number of inputs.
e.g. reduce latency by
processing several AES-CBC
messages in parallel.
Simplest if many messages
have the same length.

Number of times a key is
Most (not all) ciphers
expect precomputation of
“expanded keys”.

Warning: Do not solely com
“agility” of two ciphers.
Cipher with better “agility”
can be consistently slower.

Plaintext length and associated-data length.
Huge impact on performance.

Warning: Do not solely compare "overhead" of two ciphers. Cipher with larger "overhead" can be consistently faster.

Number of inputs.
e.g. reduce latency by processing several AES-CBC messages in parallel. Simplest if many messages have the same length.

Number of times a key is used.
Most (not all) ciphers expect precomputation of "expanded keys".

Warning: Do not solely compare "agility" of two ciphers. Cipher with better "agility" can be consistently slower.

Plaintext length and associated-data length.
Huge impact on performance.

Warning: Do not solely compare "overhead" of two ciphers. Cipher with larger "overhead" can be consistently faster.

Number of inputs.
e.g. reduce latency by processing several AES-CBC messages in parallel. Simplest if many messages have the same length.

Number of times a key is used.
Most (not all) ciphers expect precomputation of "expanded keys".

Warning: Do not solely compare "agility" of two ciphers. Cipher with better "agility" can be consistently slower.

General input scheduling.
Reduce latency by processing key and nonce before seeing associated data; associated data before plaintext.

xt length
ociated-data length.
npact on performance.

g: Do not solely compare
ad" of two ciphers.
with larger "overhead"
consistently faster.

r of inputs.
uce latency by
ng several AES-CBC
s in parallel.
t if many messages
e same length.

Number of times a key is used.
Most (not all) ciphers
expect precomputation of
"expanded keys".

Warning: Do not solely compare
"agility" of two ciphers.
Cipher with better "agility"
can be consistently slower.

General input scheduling.
Reduce latency by
processing key and nonce
before seeing associated data;
associated data before plaintext.

Schedul
schedul
Typically
to right.
processi

("Increm
Update
when inp

ata length.
erformance.

solely compare
o ciphers.
“overhead”
y faster.

s.
y by
AES-CBC
el.
messages
gth.

Number of times a key is used.
Most (not all) ciphers
expect precomputation of
“expanded keys”.

Warning: Do not solely compare
“agility” of two ciphers.
Cipher with better “agility”
can be consistently slower.

General input scheduling.
Reduce latency by
processing key and nonce
before seeing associated data;
associated data before plaintext.

Scheduling withi
scheduling withi
Typically receive d
to right. Reduce *I*
processing earlier p

(“Incrementality”:
Update output effi
when input is mod

h.

e.

pare

d”

t

Number of times a key is used.
Most (not all) ciphers expect precomputation of “expanded keys”.

Warning: Do not solely compare “agility” of two ciphers. Cipher with better “agility” can be consistently slower.

General input scheduling.
Reduce latency by processing key and nonce before seeing associated data; associated data before plaintext.

Scheduling within plaintex
scheduling within cipherte
Typically receive data from l to right. Reduce *latency* by processing earlier parts first.

(“Incrementality”: Update output efficiently when input is modified.)

Number of times a key is used.
Most (not all) ciphers expect precomputation of "expanded keys".

Warning: Do not solely compare "agility" of two ciphers. Cipher with better "agility" can be consistently slower.

General input scheduling.
Reduce latency by processing key and nonce before seeing associated data; associated data before plaintext.

Scheduling within plaintext; scheduling within ciphertext.
Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

("Incrementality": Update output efficiently when input is modified.)

Number of times a key is used. Most (not all) ciphers expect precomputation of “expanded keys”.

Warning: Do not solely compare “agility” of two ciphers. Cipher with better “agility” can be consistently slower.

General input scheduling. Reduce latency by processing key and nonce before seeing associated data; associated data before plaintext.

Scheduling within plaintext; scheduling within ciphertext. Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

(“Incrementality”: Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

r of times a key is used.

ot all) ciphers
precomputation of
ed keys".

: Do not solely compare
of two ciphers.
vith better "agility"
consistently slower.

input scheduling.

latency by
ng key and nonce
eeing associated data;
ed data before plaintext.

Scheduling within plaintext; scheduling within ciphertext. Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

("Incrementality": Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

Interme

Higher-le
long plai
each sep
⇒ small

Do bette
similar f

s a key is used.

hers

ation of

solely compare

phers.

r "agility"

y slower.

heduling.

d nonce

ciated data;

efore plaintext.

Scheduling within plaintext; scheduling within ciphertext. Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

("Incrementality": Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

Intermediate tag

Higher-level protoc

long plaintext into

each separately au

⇒ small buffer is s

Do better by integ

similar feature into

used.

pare

a;

text.

Scheduling within plaintext; scheduling within ciphertext.
Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

(“Incrementality”: Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticate
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Scheduling within plaintext; scheduling within ciphertext.
Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

("Incrementality": Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated. ⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Scheduling within plaintext; scheduling within ciphertext.

Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

(“Incrementality”: Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

Intermediate tags.

Higher-level protocol splits long plaintext into packets, each separately authenticated. ⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.

Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Scheduling within plaintext; scheduling within ciphertext.
Typically receive data from left to right. Reduce *latency* by processing earlier parts first.

("Incrementality": Update output efficiently when input is modified.)

Also save *area* if large plaintext does not need large buffer. Warning: Large ciphertext needs large buffer or analysis of security impact of releasing unverified plaintext.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated. ⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

ding within plaintext;
ing within ciphertext.
y receive data from left
Reduce *latency* by
ng earlier parts first.

nentality":
output efficiently
put is modified.)

e *area* if large plaintext
t need large buffer.
g: Large ciphertext
rge buffer or
of security impact of
g unverified plaintext.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

Support

Simplici
Cryptana
that are

n plaintext;
n ciphertext.
lata from left
atency by
parts first.

iciently
dified.)

arge plaintext
ge buffer.
phertext
or
y impact of
d plaintext.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

Support for crypta

Simplicity.
Cryptanalysts prio
that are easy to u

xt;

ext.

left

ext

of

t.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

Support for cryptanalysis

Simplicity.
Cryptanalysts prioritize targe
that are easy to understand.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

Support for cryptanalysis

Simplicity.
Cryptanalysts prioritize targets that are easy to understand.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

Support for cryptanalysis

Simplicity.
Cryptanalysts prioritize targets that are easy to understand.

Scalability.
Reduced-round targets, reduced-word targets, etc.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.
⇒ small buffer is safe.

Do better by integrating similar feature into cipher?

Other operations.
Single circuit for, e.g., hash and authenticated cipher; for different key sizes; etc.

Cache context.
How well does the system fit into fast memory?

Support for cryptanalysis

Simplicity.
Cryptanalysts prioritize targets that are easy to understand.

Scalability.
Reduced-round targets, reduced-word targets, etc.

Proofs.
The phrase “proof of security” is almost always fraudulent.
Proof says that attacks *meeting certain constraints* are difficult, or *as difficult as another problem*.
Can be useful for cryptanalysts.