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

sender
\[m\]
\[c \leftarrow k\]

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

\[ m \]

\[ c \leftarrow k \]

network

\[ k: \text{ secret key.} \]
\[ m: \text{ variable-length plaintext.} \]
\[ c: \text{ variable-length ciphertext.} \]

Authenticated encryption

sender

\[ m \]

\[ c \leftarrow k \]

network

\[ k: \text{ secret key.} \]
\[ m: \text{ variable-length plaintext.} \]
\[ c: \text{ variable-length ciphertext.} \]
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

sender
↓
\[ m \]
↓
\[ c \leftarrow k \]

network

\( k \): secret key.
\( m \): variable-length plaintext.
\( c \): variable-length ciphertext.

Authenticated encryption

sender
↓
\[ m \]
↓
\[ c \leftarrow k \]

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
→
\( m \)
→
\( c \leftarrow k \)
→
network

\( k \): secret key.
\( m \): variable-length plaintext.
\( c \): variable-length ciphertext.

Authenticated encryption

sender
→
\( m \)
→
\( c \leftarrow k \)
→
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”. 
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”.

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**
  - \( m \)
  - \( c \leftarrow k \)
- **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.
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

$n$: 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.

Associated data

sender
↓ ↓ ↓ ↓

n
↓ ↓

m
↓ ↓
a
↓ ↓
c
↓ ↓
k
← ←
network

k: secret key.
n: public message number.
a: variable-length associated data.
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".

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 associated data.
m: variable-length plaintext.
c: variable-length ciphertext.
Authenticated encryption

\[
\begin{array}{c}
\text{sender} \\
\downarrow \\
m \\
\downarrow \\
c \\
\downarrow \\
k \\
\leftarrow \\
network
\end{array}
\]

\[k: \text{secret key.}\]
\[n: \text{public message number.}\]
\[m: \text{variable-length plaintext.}\]
\[c: \text{variable-length ciphertext.}\]

Changes in message number hide repetitions of plaintext.

Associated data

\[
\begin{array}{c}
\text{sender} \\
\downarrow \\
n \\
\downarrow \\
m \\
\downarrow \\
a \\
\downarrow \\
c \\
\downarrow \\
k \\
\leftarrow \\
network
\end{array}
\]

\[k: \text{secret key.}\]
\[n: \text{public message number.}\]
\[a: \text{variable-length associated data.}\]
\[m: \text{variable-length plaintext.}\]
\[c: \text{variable-length ciphertext.}\]
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

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.

$m$: variable-length plaintext.

$c$: variable-length ciphertext.

$a$: variable-length associated data.

Changes in message number hide repetitions of plaintext.

No problem repeating $a$. 
Sender

↓

m

↓

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

↓

k

network

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

sender

↓

n

↓

m

↓

a

↓

k

network

k: secret key.
n: secret message number.
a: variable-length associated data.
m: variable-length plaintext.
c: variable-length ciphertext.
Associated data

- **$n$**: public message number.
- **$m$**: variable-length plaintext.
- **$a$**: variable-length associated data.
- **$c$**: variable-length ciphertext.

$\kappa$: secret key.

No problem repeating $a$.

Secret message numbers

- **$n$**: secret message number.
- **$m$**: variable-length plaintext.
- **$a$**: variable-length associated data.
- **$c$**: variable-length ciphertext.

$\kappa$: secret key.
**Associated data**

- **sender**
- **network**

- \( k \): secret key.
- \( n \): public message number.
- \( m \): variable-length plaintext.
- \( a \): variable-length associated data.
- \( c \): variable-length ciphertext.

No problem repeating \( a \).

**Secret message numbers**

- **sender**
- **network**

- \( k \): secret key.
- \( n \): secret message number.
- \( m \): variable-length plaintext.
- \( a \): variable-length associated data.
- \( c \): 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.
In the associated data model:

- $k$: secret key.
- $n$: public message number.
- $a$: variable-length associated data.
- $m$: variable-length plaintext.
- $c$: variable-length ciphertext.

The associated data repeat issue:

- No problem repeating $a$.

In the secret message numbers model:

- $k$: 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.
Secret message numbers

| sender | n | m | a | c | k | network |

\(k\): 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.
Secret message numbers

sender

n \[ \rightarrow \]\ n
↓ ↓ ↓ ↓ ↓ ↓ 
m
↓ ↓ 
a
↓ ↓ 
↓ ↓ 
c
↓ ↓ 
k
← ← 
network

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.
Secret message numbers

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.

\(k\): 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.

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

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

Secret key.
Secret message number.
Variable-length associated data.
Variable-length plaintext.
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.

**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**.
Forges \((n, m, a)\) that receiver accepts but that legitimate sender never encrypted.

“**INT-PTXT**” 
(integrity of plaintexts) means protection against such attacks.

Stronger goal:
Forges at least \(f\) messages.

**Ciphertext corruption.**
Forges \(c\) 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.
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?
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?

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

Replay.

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

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

**Replay:**

Convince receiver accepts but that legitimate sender never produced.
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.
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.
Forger $c$ 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?
Ciphertext corruption.
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?

Plaintext espionage.
Figure out user’s secret message.
Ciphertext corruption.

This occurs when a malicious third party attempts to create a ciphertext (c) that the receiver accepts but that the 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?

Plaintext espionage.

Figure out user’s secret message.
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?

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.
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.
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.

What are the attacker’s resources?
Extensive computation.

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

What are the attacker’s resources?
Extensive computation.
Are 80-bit keys adequate?
Are 128-bit keys adequate?
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.

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?
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.
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.

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.
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.
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?
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?

**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.
What are the attacker’s resources?

- Extensive computation.

Are 80-bit keys adequate?

- No.

Are 128-bit keys adequate?

- No.

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?

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.
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.
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.

Many messages.

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

Other designers argue that eliminating such requirements adds robustness.
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.
Some attackers carry out attacks that are feasible but not economically rational.

3. 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.

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.
Back-of-the-envelope figures:

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

2^{44} \text{ watts}: \text{ world power usage.}

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

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

**Scalable quantum computers.**

2^{64} \text{ 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} \text{ 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.
Back-of-the-envelope figures:

- 257 watts: received by Earth’s atmosphere from the Sun.
- 244 watts: world power usage.
- 226 watts: one computer center costing 230 dollars.

1 watt: power for 268 bit operations per year using mass-market GPUs.

Scalable quantum computers.

- 264 simple quantum operations to find a 128-bit key using Grover’s algorithm.

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

Some designers blame the user: “switch keys after 220 messages.” Other designers argue that eliminating such requirements adds robustness.

Consensus: Unacceptable to blame the user.

Chosen plaintexts, chosen ciphertexts, chosen message numbers.

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

Many messages.
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 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?
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 $n'$. 
Many messages.

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

plaintexts,
ciphertexts,
message numbers.

Thus:

Notable to blame the user.

Messages 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 $n'$.

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 messages. Some designers blame the user: “switch keys after 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 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 $n'$.

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

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?
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?

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).
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?

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).
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?

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).
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).
Software side channels.

- secret branches,
- secret memory addresses.
- On some CPUs, secret multiplication inputs.

Hardware side channels.

- Power consumption,
- Electromagnetic radiation, etc.
- 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).

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.

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

Secret keys.
<table>
<thead>
<tr>
<th>What performance is measured?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical performance metrics</td>
</tr>
<tr>
<td>for ASICs:</td>
</tr>
<tr>
<td>Low energy (joules) per byte.</td>
</tr>
<tr>
<td>Low power (watts).</td>
</tr>
<tr>
<td>Low area (square micrometers;</td>
</tr>
<tr>
<td>loosely predicted by</td>
</tr>
<tr>
<td>“gate equivalents”).</td>
</tr>
<tr>
<td>High throughput</td>
</tr>
<tr>
<td>(bytes per second).</td>
</tr>
<tr>
<td>Low latency (seconds;</td>
</tr>
<tr>
<td>very loosely predicted by</td>
</tr>
<tr>
<td>cycles).</td>
</tr>
</tbody>
</table>

| 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.
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.
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.

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.

Send valid data, receive valid data, or receive invalid data?

“Encrypt then MAC” skips cost of decryption for forgeries.
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.

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.
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.

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.
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.
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.

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.
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.

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.
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.

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.
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.
Operations are measured?
Authenticate only, or encrypt and authenticate?
Cost per byte of a can be far below cost per byte of m.
Valid data, receive valid data, or receive invalid data?
"Encrypt then MAC" skips decryption for forgeries.
Different area targets:
encryption/authentication circuit;
verification/decryption circuit;
circuit that can dynamically select either operation.

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.
What operations are measured? Authenticate only, or encrypt and authenticate?

Cost per byte of $a$ can be far below cost per byte of $m$.

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.


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.
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.

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.

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.

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.
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. Reduce latency by processing several AES-CBC messages in parallel. Easiest 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.

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.)
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.)
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.
Number of times a key is used.

Most (not all) ciphers expect precomputation of "expanded keys".

Warning: Do not solely compare of two ciphers.

with better “agility” consistently slower.

**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.

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?
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.

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

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

Reduce area if large plaintext does not need large buffer. ⇒ small buffer is safe.

Intermediate tags.
Higher-level protocol splits long plaintext into packets, each separately authenticated.

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.
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.)

Large 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?
**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. \(\Rightarrow\) 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.