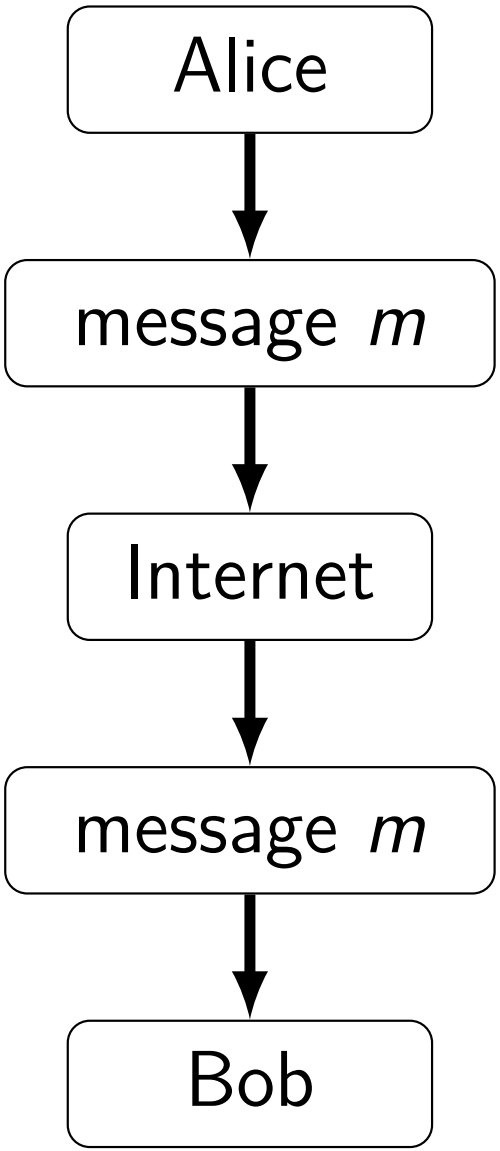
Symmetric cryptanalysis

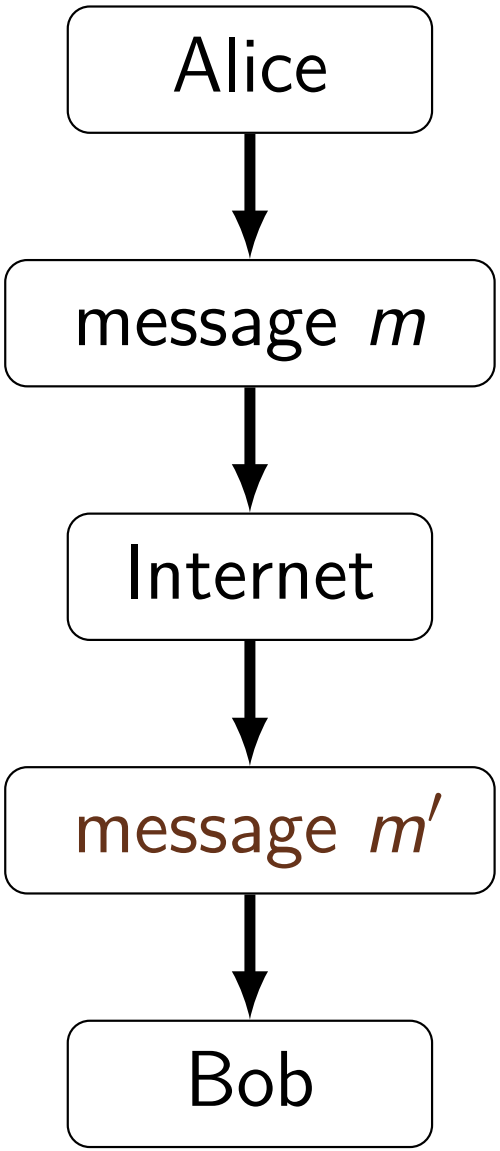
Daniel J. Bernstein

Attack 0: look for plaintext



Attacker sees message m as
m travels through the Internet.
This is a failure of **confidentiality**.

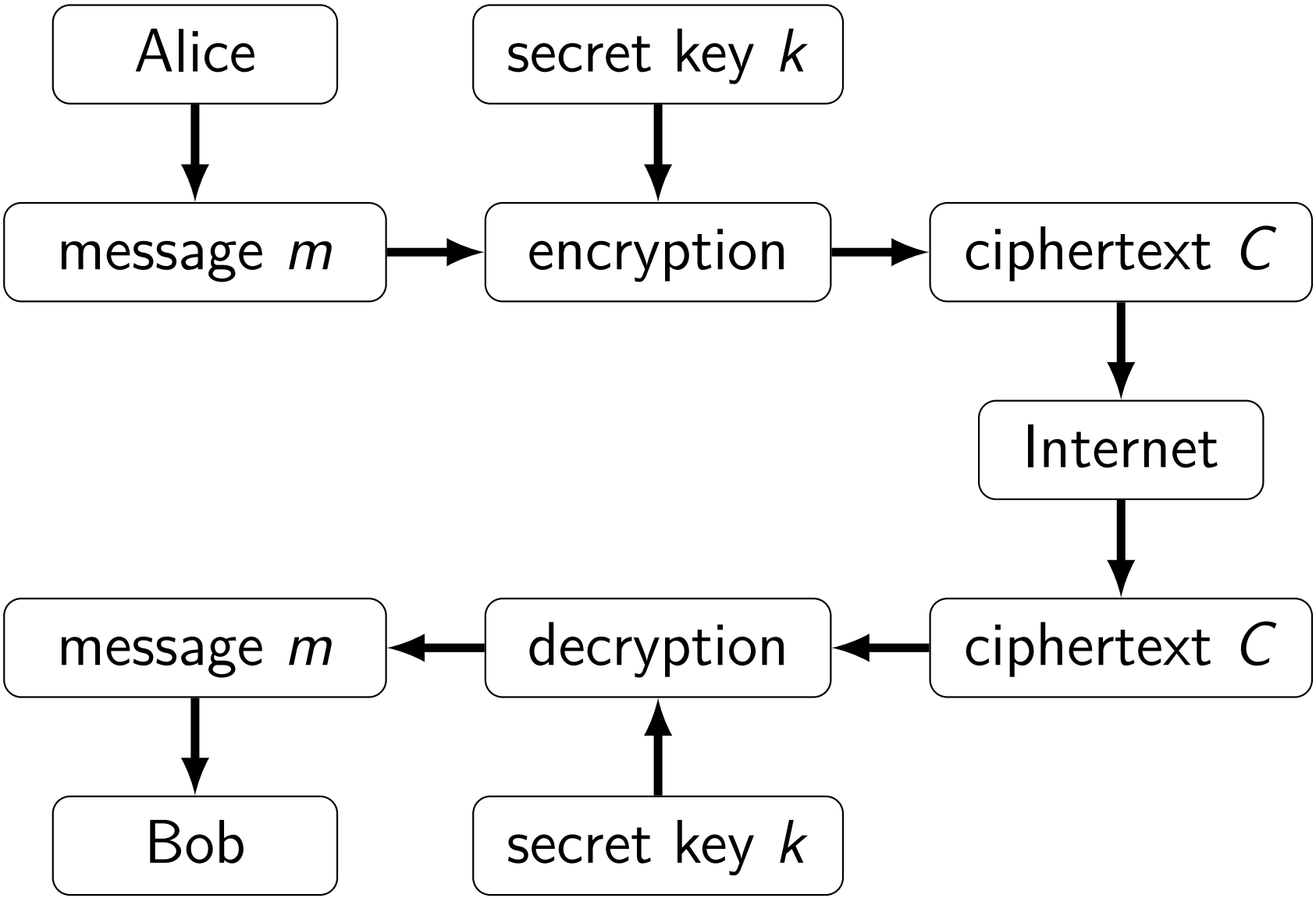
Attack 0: look for plaintext



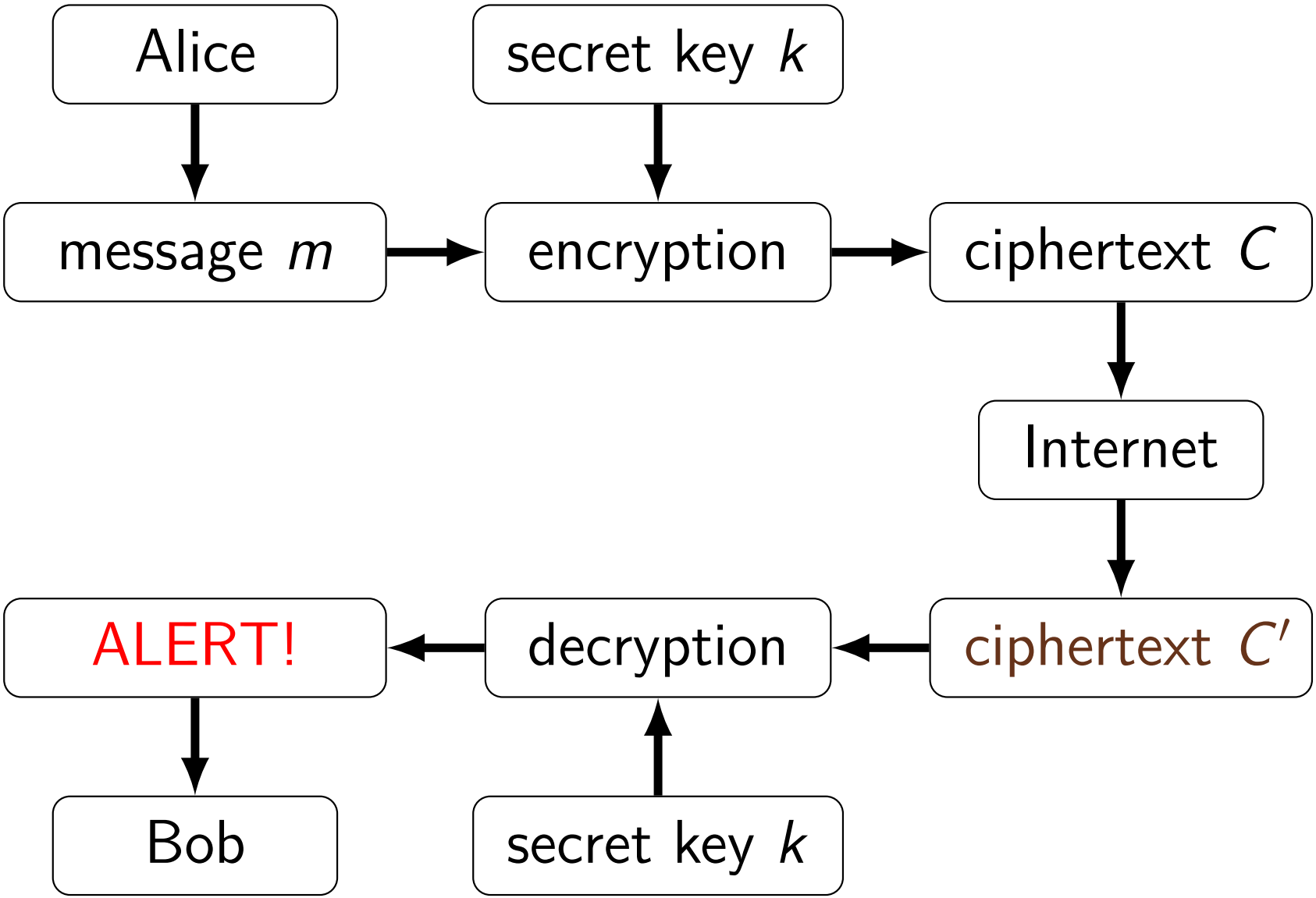
Attacker sees message m as
m travels through the Internet.
This is a failure of **confidentiality**.

Can also carry out active attacks:
modify message m into m',
or forge a new message m'.
Bob thinks m' comes from Alice.
This is a failure of **integrity**.

Symmetric cryptography: basic data flow



Symmetric cryptography: basic data flow



Information-theoretic security

The one-time pad

Say Alice's first message m has 30 digits.

Alice's encryption: $C = (m + k_1) \bmod 10^{30}$
where $k_1 = k \bmod 10^{30}$.

The one-time pad

Say Alice's first message m has 30 digits.

Alice's encryption: $C = (m + k_1) \bmod 10^{30}$
where $k_1 = k \bmod 10^{30}$.

If k_1 is uniform random (i.e., all choices of k_1 between 0 and $10^{30} - 1$ are equally likely) then attacker seeing C has no idea what m is.

The one-time pad

Say Alice's first message m has 30 digits.

Alice's encryption: $C = (m + k_1) \bmod 10^{30}$
where $k_1 = k \bmod 10^{30}$.

If k_1 is uniform random (i.e., all choices of k_1 between 0 and $10^{30} - 1$ are equally likely) then attacker seeing C has no idea what m is.

Bob's decryption: $m = (C - k_1) \bmod 10^{30}$.

The one-time pad

Say Alice's first message m has 30 digits.

Alice's encryption: $C = (m + k_1) \bmod 10^{30}$
where $k_1 = k \bmod 10^{30}$.

If k_1 is uniform random (i.e., all choices of k_1
between 0 and $10^{30} - 1$ are equally likely)
then attacker seeing C has no idea what m is.

Bob's decryption: $m = (C - k_1) \bmod 10^{30}$.

"One-time": Alice+Bob burn k_1 after it is used;
i.e., Alice+Bob replace k with $\lfloor k/10^{30} \rfloor$.
Next message uses next digits k_2 from k.

Looking for plaintext, revisited

Attacker sees Alice sending *some* 30-digit message to Bob. This is often an important leak of private information, even though the digits are hidden.

Looking for plaintext, revisited

Attacker sees Alice sending *some* 30-digit message to Bob. This is often an important leak of private information, even though the digits are hidden.

This is a failure to protect confidentiality of **metadata**: sender identity, receiver identity, message length, message timing.

Looking for plaintext, revisited

Attacker sees Alice sending *some* 30-digit message to Bob. This is often an important leak of private information, even though the digits are hidden.

This is a failure to protect confidentiality of **metadata**: sender identity, receiver identity, message length, message timing.

Example of technique to try to hide metadata: "mixnets". Won't say more about this today.

Message-authentication codes (MACs)

The one-time pad doesn't stop attacker from modifying messages.
e.g. attacker adds 1 to C; this adds 1 to m.

Message authentication protects integrity:

- Alice uses more digits of secret key to compute an **authenticator** of C.
- Alice sends the authenticator along with C.
- Bob recomputes the authenticator.
- Bob rejects C if authenticator doesn't match.

A simple MAC

Standardize a prime $p = 1000003$.

To authenticate a 30-digit message
$(C_1, C_2, C_3, C_4, C_5)$
where each $C_i \in \{0, 1, \ldots, 999999\}$,
Alice sends 6-digit authenticator
$(((C_1 r_1 + \cdots + C_5 r_5) \bmod p) + s) \bmod 1000000$,
using 36 digits from secret key:
$r_1, r_2, r_3, r_4, r_5, s \in \{0, 1, \ldots, 999999\}$.

Example 1

$r_1 = 314159$, $r_2 = 265358$, $r_3 = 979323$,
$r_4 = 846264$, $r_5 = 338327$, $s = 950288$,
$C = 000006\,000007\,000000\,000000\,000000$:

Example 1

$r_1 = 314159$, $r_2 = 265358$, $r_3 = 979323$,
$r_4 = 846264$, $r_5 = 338327$, $s = 950288$,
$C = 000006\,000007\,000000\,000000\,000000$:

Alice computes authenticator
$(6r_1 + 7r_2 \bmod p) + s \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003) + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 = 692739.$

Example 1

$r_1 = 314159$, $r_2 = 265358$, $r_3 = 979323$,
$r_4 = 846264$, $r_5 = 338327$, $s = 950288$,
$C = 000006\,000007\,000000\,000000\,000000$:

Alice computes authenticator
$(6r_1 + 7r_2 \bmod p) + s \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003) + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 = 692739$.

Alice transmits
$000006\,000007\,000000\,000000\,000000\,692739$.

A MAC using fewer secret digits

Alice sends 6-digit authenticator
$(C_1 r + C_2 r^2 + \cdots + C_5 r^5 \bmod p) + s \bmod 1000000$,
using 12 digits from secret key:
$r, s \in \{0, 1, \ldots, 999999\}$.

i.e.: take $r_i = r^i$ in previous authenticator,
replacing separate 6-digit secrets $r_1, r_2, \ldots$
with just one 6-digit secret r.

Example 2

$r = 314159$, $s = 265358$,
$C = 000006\,000007\,000000\,000000\,000000$:

Example 2

$r = 314159$, $s = 265358$,
$C = 000006\,000007\,000000\,000000\,000000$:

Sender computes authenticator
$(6r + 7r^2 \bmod p) + s \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\quad \bmod 1000003) + 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 = 218669.$

Example 2

$r = 314159$, $s = 265358$,
$C = 000006\,000007\,000000\,000000\,000000$:

Sender computes authenticator
$(6r + 7r^2 \bmod p) + s \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\quad \bmod 1000003) + 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 = 218669$.

Sender transmits authenticated message
$000006\,000007\,000000\,000000\,000000\,218669$.

Is this secure?

Attacker sees C, a.

Attacker's goal: Find C', a' such that

- $a' = (C'(r) \bmod p) + s \bmod 1000000$ and
- $C' \neq C$.

Here $C'(r) = C'_1 r + C'_2 r^2 + C'_3 r^3 + C'_4 r^4 + C'_5 r^5$.

Is this secure?

Attacker sees C, a.

Attacker's goal: Find C', a' such that

- $a' = (C'(r) \bmod p) + s \bmod 1000000$ and
- $C' \neq C$.

Here $C'(r) = C'_1 r + C'_2 r^2 + C'_3 r^3 + C'_4 r^4 + C'_5 r^5$.

Obvious attack: Choose any $C' \neq C$. Choose uniform random a'. Success chance $1/1000000$.

Is this secure?

Attacker sees C, a.

Attacker's goal: Find C', a' such that

- $a' = (C'(r) \bmod p) + s \bmod 1000000$ and
- $C' \neq C$.

Here $C'(r) = C'_1 r + C'_2 r^2 + C'_3 r^3 + C'_4 r^4 + C'_5 r^5$.

Obvious attack: Choose any $C' \neq C$. Choose uniform random a'. Success chance $1/1000000$.

Can repeat attack. Each forgery has chance $1/1000000$ of being accepted.

A more powerful attack

Choose $a' = a$. Choose $C' \neq C$ so that the polynomial $C'(x) - C(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p.

A more powerful attack

Choose $a' = a$. Choose $C' \neq C$ so that the polynomial $C'(x) - C(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p.

e.g. $C = (100, 0, 0, 0, 0)$, $C' = (125, 1, 0, 0, 1)$: $C'(x) - C(x) = x^5 + x^2 + 25x$ which has five roots mod p, namely $0, 299012, 334447, 631403, 735144$.

A more powerful attack

Choose $a' = a$. Choose $C' \neq C$ so that the polynomial $C'(x) - C(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p.

e.g. $C = (100, 0, 0, 0, 0)$, $C' = (125, 1, 0, 0, 1)$: $C'(x) - C(x) = x^5 + x^2 + 25x$ which has five roots mod p, namely $0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Same attack can be even more powerful

Success chance can be above $5/1000000$.

Same attack can be even more powerful

Success chance can be above $5/1000000$.

Example: If $C(334885) \bmod p$
$\in \{1000000, 1000001, 1000002\}$
then a forgery (C', a') with
$C'(x) = C(x) + x^5 + x^2 + 25x$ and $a' = a$
succeeds not just for
$r \in \{0, 299012, 334447, 631403, 735144\}$
from previous slide, but also for $r = 334885$,
which is a root of $C'(x) - C(x) + 1000000$.

Attacker can't do much better than this

Every choice of (C', a') with $C' \neq C$ has chance $\leq 15/1000000$ of being accepted by receiver.

Attacker can't do much better than this

Every choice of (C', a') with $C' \neq C$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: the polynomial
$(C'(x) - C(x) - a' + a) \cdot$
$(C'(x) - C(x) - a' + a + 10^6) \cdot$
$(C'(x) - C(x) - a' + a - 10^6)$
has at most 15 roots modulo p.

Attacker can't do much better than this

Every choice of (C', a') with $C' \neq C$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: the polynomial
$(C'(x) - C(x) - a' + a) \cdot$
$(C'(x) - C(x) - a' + a + 10^6) \cdot$
$(C'(x) - C(x) - a' + a - 10^6)$
has at most 15 roots modulo p.

Warning: very easy to break the simpler variant
$(C_1 + C_2 r + \cdots + C_5 r^4 \bmod p) + s \bmod 1000000$;
simply solve $C'(x) - C(x) = a' - a$.

Scale up for serious security

Poly1305 is a MAC that uses 128-bit r's,
with 22 bits cleared for speed. Adds s mod 2^{128}.

Scale up for serious security

Poly1305 is a MAC that uses 128-bit r's, with 22 bits cleared for speed. Adds s mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16\rceil$ choices of r.

Scale up for serious security

Poly1305 is a MAC that uses 128-bit r's, with 22 bits cleared for speed. Adds s mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16\rceil$ choices of r.

D forgeries are all rejected with probability $\geq 1 - 8D\lceil L/16\rceil/2^{106}$.

Scale up for serious security

Poly1305 is a MAC that uses 128-bit r's, with 22 bits cleared for speed. Adds s mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8\lceil L/16\rceil$ choices of r.

D forgeries are all rejected with probability $\geq 1 - 8D\lceil L/16\rceil/2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$:
Pr[all rejected] ≥ 0.9999999998.

More variations

Alice and Bob can reuse r for many messages.
But important to use new s for each message.

More variations

Alice and Bob can reuse r for many messages.
But important to use new s for each message.

Alice and Bob can use variable-length messages,
as long as different messages correspond to
different polynomials mod p.

More variations

Alice and Bob can reuse r for many messages. But important to use new s for each message.

Alice and Bob can use variable-length messages, as long as different messages correspond to different polynomials mod p.

e.g. Poly1305: Split string into 16-byte chunks, maybe with smaller final chunk; append 1 to each chunk; view as little-endian integers in $\{1, 2, 3, \ldots, 2^{129}\}$. Multiply first chunk by r, add next chunk, multiply by r, etc., add last chunk, multiply by r, mod $2^{130} - 5$, add s mod 2^{128}.

Is this safe in the real world?

Maybe the secret key isn't truly random. Example: SmartFacts exploited bad randomness-generation hardware in smart cards that had NIST/CSE FIPS certification and BSI Common Criteria certification.

Is this safe in the real world?

Maybe the secret key isn't truly random. Example: SmartFacts exploited bad randomness-generation hardware in smart cards that had NIST/CSE FIPS certification and BSI Common Criteria certification.

Maybe the secrets are leaked, for example through variations in timings of computations. See, e.g., "potentially determine authentication tag validity via timing analysis" in CVE-2026-3337 (AWS-LC).

Is this safe in the real world?

Maybe the secret key isn't truly random. Example: SmartFacts exploited bad randomness-generation hardware in smart cards that had NIST/CSE FIPS certification and BSI Common Criteria certification.

Maybe the secrets are leaked, for example through variations in timings of computations. See, e.g., "potentially determine authentication tag validity via timing analysis" in CVE-2026-3337 (AWS-LC).

Maybe an attacker corrupted the random-number generator. See, e.g., NSA's backdoored Dual EC.

Symmetric vs. public-key

Spying on the initial key exchange

Even if Alice generates a truly random k,
she still has to share it with Bob.

If an attacker sees this communication of k,
all security is lost.

Spying on the initial key exchange

Even if Alice generates a truly random k,
she still has to share it with Bob.

If an attacker sees this communication of k,
all security is lost.

Public-key encryption tries to solve this,
allowing Alice and Bob to share secret k
through a public communication channel.
Many proposed public-key cryptosystems are broken,
but details are outside today's scope.

Should users eliminate symmetric crypto?

Many public-key cryptosystems can directly encrypt user messages. But this turns out to open up many more attack possibilities.

Should users eliminate symmetric crypto?

Many public-key cryptosystems can directly encrypt user messages. But this turns out to open up many more attack possibilities.

Less dangerous: use public-key crypto to share a random key; then rely on symmetric crypto for user messages. Using symmetric crypto creates some risks, but avoids bigger risks in public-key crypto.

Should users eliminate symmetric crypto?

Many public-key cryptosystems can directly encrypt user messages. But this turns out to open up many more attack possibilities.

Less dangerous: use public-key crypto to share a random key; then rely on symmetric crypto for user messages. Using symmetric crypto creates some risks, but avoids bigger risks in public-key crypto.

This split also means that a single public-key operation handles any number of messages from Alice to Bob, so less incentive for developers to scale down public-key security levels.

Ciphers

The key-size mismatch

Public-key crypto typically shares a short secret key: e.g., Alice and Bob end up sharing a 256-bit k.

The one-time pad uses 1MB of secrets to encrypt 1MB of user data. Also, a polynomial MAC needs a new s for each message. How do Alice and Bob share all this secret data?

Using ciphers to expand keys

AES expands 256-bit secret k into many secrets $F(k,1), F(k,2), F(k,3), \ldots$; AES-GCM uses these for the one-time pad and a polynomial MAC.

Using ciphers to expand keys

AES expands 256-bit secret k into many secrets $F(k,1), F(k,2), F(k,3), \ldots$; AES-GCM uses these for the one-time pad and a polynomial MAC.

ChaCha expands 256-bit secret k into many secrets $F(k,1), F(k,2), F(k,3), \ldots$; ChaCha20-Poly1305 uses these for the one-time pad and a polynomial MAC. (Not the same function F as for AES.)

Using ciphers to expand keys

AES expands 256-bit secret k into many secrets $F(k,1), F(k,2), F(k,3), \ldots$; AES-GCM uses these for the one-time pad and a polynomial MAC.

ChaCha expands 256-bit secret k into many secrets $F(k,1), F(k,2), F(k,3), \ldots$; ChaCha20-Poly1305 uses these for the one-time pad and a polynomial MAC. (Not the same function F as for AES.)

Definition of **PRG** ("pseudorandom generator"): Attacker can't distinguish $F(k,1), F(k,2), F(k,3), \ldots$ from string of independent uniform random blocks.

PRF, PRP

PRF ("pseudorandom function"):
Attacker can't distinguish
$F(k,1), F(k,2), F(k,3), \ldots$ from independent uniform random blocks, given access to a server that returns $F(k,i)$ given i. Server is called an **oracle**.

PRF, PRP

PRF ("pseudorandom function"):
Attacker can't distinguish
$F(k,1), F(k,2), F(k,3), \ldots$ from independent uniform random blocks, given access to a server that returns $F(k,i)$ given i. Server is called an **oracle**.

PRP ("pseudorandom permutation"):
Attacker can't distinguish
$F(k,1), F(k,2), F(k,3), \ldots$ from independent uniform random **distinct** blocks, given oracle.

PRF, PRP

PRF ("pseudorandom function"):
Attacker can't distinguish
$F(k,1), F(k,2), F(k,3), \ldots$ from independent uniform random blocks, given access to a server that returns $F(k,i)$ given i. Server is called an **oracle**.

PRP ("pseudorandom permutation"):
Attacker can't distinguish
$F(k,1), F(k,2), F(k,3), \ldots$ from independent uniform random **distinct** blocks, given oracle.

If block size is big then PRP $\Rightarrow$ PRF $\Rightarrow$ PRG.

The importance of block sizes

Small block sizes are dangerous.
PRF property fails; often application security fails.

The importance of block sizes

Small block sizes are dangerous.
PRF property fails; often application security fails.

e.g. 2016 Bhargavan–Leurent `sweet32.info`:
Triple-DES broken in TLS. Same attack also breaks small block sizes in NSA's Simon, Speck.

The importance of block sizes

Small block sizes are dangerous.
PRF property fails; often application security fails.

e.g. 2016 Bhargavan–Leurent `sweet32.info`:
Triple-DES broken in TLS. Same attack also breaks small block sizes in NSA's Simon, Speck.

AES block size: 128 bits. PRF attack chance $\approx q^2/2^{129}$ if AES is used for q blocks. Is this safe? How big is q?

The importance of block sizes

Small block sizes are dangerous.
PRF property fails; often application security fails.

e.g. 2016 Bhargavan–Leurent `sweet32.info`:
Triple-DES broken in TLS. Same attack also breaks small block sizes in NSA's Simon, Speck.

AES block size: 128 bits. PRF attack chance $\approx q^2/2^{129}$ if AES is used for q blocks. Is this safe? How big is q?

ChaCha20 block size: 512 bits.

Modes

Can prove confidentiality and integrity
of AES-GCM and ChaCha20-Poly1305
assuming AES and ChaCha20 are PRFs.

Modes

Can prove confidentiality and integrity of AES-GCM and ChaCha20-Poly1305 *assuming* AES and ChaCha20 are PRFs.

Generalization: Prove security of $M(F)$ *assuming* cipher F is a PRF.

Terminology: M is a **mode of use** (aka **mode of operation**) of F.

Modes

Can prove confidentiality and integrity
of AES-GCM and ChaCha20-Poly1305
assuming AES and ChaCha20 are PRFs.

Generalization: Prove security of $M(F)$
assuming cipher F is a PRF.

Terminology: M is a **mode of use**
(aka **mode of operation**) of F.

Check what security features a mode is promising:
e.g., CTR ("counter mode") is good for encryption
but does not try to provide authentication.

Some “security proofs” are wrong

2004 Rogaway claimed a “proof of security” for OCB2 mode. 2009: OCB2 was standardized.

Some “security proofs” are wrong

2004 Rogaway claimed a “proof of security” for OCB2 mode. 2009: OCB2 was standardized. 2019 Inoue–Iwata–Minematsu–Poettering broke OCB2.

Some “security proofs” are wrong

2004 Rogaway claimed a “proof of security” for OCB2 mode. 2009: OCB2 was standardized. 2019 Inoue–Iwata–Minematsu–Poettering broke OCB2.

2007 McGrew–Fluhrer claimed a “proof of security” for XCB mode. 2010: XCB was standardized.

Some "security proofs" are wrong

2004 Rogaway claimed a "proof of security" for OCB2 mode. 2009: OCB2 was standardized. 2019 Inoue–Iwata–Minematsu–Poettering broke OCB2.

2007 McGrew–Fluhrer claimed a "proof of security" for XCB mode. 2010: XCB was standardized. 2015 Chakraborty–Hernandez-Jimenez–Sarkar broke XCB for unusual message lengths but claimed a "proof of security" for XCB for normal lengths.

Some "security proofs" are wrong

2004 Rogaway claimed a "proof of security" for OCB2 mode. 2009: OCB2 was standardized. 2019 Inoue–Iwata–Minematsu–Poettering broke OCB2.

2007 McGrew–Fluhrer claimed a "proof of security" for XCB mode. 2010: XCB was standardized. 2015 Chakraborty–Hernandez-Jimenez–Sarkar broke XCB for unusual message lengths but claimed a "proof of security" for XCB for normal lengths. 2025 Bhati–Andreeva broke XCB for normal lengths.

More on proofs

See Koblitz–Menezes for survey of many more flaws in "security proofs".

More on proofs

See Koblitz–Menezes for survey of many more flaws in "security proofs".

For comparison, 2021 Katz–Lindell textbook says "Proofs of security give an iron-clad guarantee—relative to the definition and assumptions—that no attacker will succeed".

More on proofs

See Koblitz–Menezes for survey of many more flaws in "security proofs".

For comparison, 2021 Katz–Lindell textbook says "Proofs of security give an iron-clad guarantee—relative to the definition and assumptions—that no attacker will succeed".

Open question: What % of "proofs" are wrong?

Sometimes ciphers are not secure

Take a correct proof that $M(F)$ is secure *assuming* F is a PRF. How do we know that AES and ChaCha20 are PRFs?

Sometimes ciphers are not secure

Take a correct proof that $M(F)$ is secure *assuming* F is a PRF. How do we know that AES and ChaCha20 are PRFs? Answer: We don't! People *conjecture* security after many failed attack efforts.

Sometimes ciphers are not secure

Take a correct proof that $M(F)$ is secure *assuming* F is a PRF. How do we know that AES and ChaCha20 are PRFs? Answer: We don't! People *conjecture* security after many failed attack efforts.

Remaining slides today:

- Simple example of block cipher: 1994 "TEA, a tiny encryption algorithm" (Wheeler–Needham). Seems to be a good cipher, *except* that block size is too small: good only for PRP, not PRF.
- Variants of this block cipher that look similar but can be quickly broken.

TEA

Notation

`uint32`: 32 bits $(b_0, b_1, \ldots, b_{31})$ representing the "unsigned" integer $b_0 + 2b_1 + \cdots + 2^{31}b_{31}$.

`+`: addition mod 2^{32}.

`c += d`: same as `c = c + d`.

`^`: xor; $\oplus$; addition of each bit separately mod 2. Lower precedence than + in C.

`<<4`: mult by 16, i.e., $(0, 0, 0, 0, b_0, b_1, \ldots, b_{27})$.

`>>5`: div by 32, i.e., $(b_5, b_6, \ldots, b_{31}, 0, 0, 0, 0, 0)$.

TEA

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Functionality

TEA is a **64-bit block cipher** with a **128-bit key**.

Functionality

TEA is a **64-bit block cipher** with a **128-bit key**.

Input: 128-bit key (`k[0]`,`k[1]`,`k[2]`,`k[3]`);
64-bit **plaintext** (`b[0]`,`b[1]`).

Output: 64-bit **ciphertext** (final `b[0]`,`b[1]`).

Functionality

TEA is a **64-bit block cipher** with a **128-bit key**.

Input: 128-bit key (`k[0]`,`k[1]`,`k[2]`,`k[3]`);
64-bit **plaintext** (`b[0]`,`b[1]`).

Output: 64-bit **ciphertext** (final `b[0]`,`b[1]`).

Can quickly **encrypt**: (key, plaintext) $\mapsto$ ciphertext.

Can quickly **decrypt**: (key, ciphertext) $\mapsto$ plaintext.

Wait, how can we decrypt?

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Answer: Each step is invertible

```
void decrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 32 * 0x9e3779b9;
  for (r = 0;r < 32;r += 1) {
    y -= x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
    x -= y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    c -= 0x9e3779b9;
  }
  b[0] = x; b[1] = y;
}
```

Generalization: "Feistel network"

(used in, e.g., 1973 Feistel–Coppersmith "Lucifer")

```
x += function1(y,k); y += function2(x,k);
x += function3(y,k); y += function4(x,k);
...
```

Decryption, inverting each step:

```
...
y -= function4(x,k); x -= function3(y,k);
y -= function2(x,k); x -= function1(y,k);
```

Do we need decryption?

If F is used only to expand key k into $F(k,1), F(k,2), \ldots$ then no need to invert F: e.g., encrypt block by adding $F(k,1)$, decrypt block by subtracting $F(k,1)$.

But some modes apply F to blocks of user data, and need F^{-1} for decryption. Literature sometimes claims that these modes have advantages.

TEA again

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

XORTEA: a bad cipher

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x ^= y^c ^ (y<<4)^k[0] ^ (y>>5)^k[1];
    y ^= x^c ^ (x<<4)^k[2] ^ (x>>5)^k[3];
  }
  b[0] = x; b[1] = y;
}
```

XORTEA vs. TEA

"Hardware-friendlier" cipher,
since xor circuit is cheaper than add.

XORTEA vs. TEA

"Hardware-friendlier" cipher,
since xor circuit is cheaper than add.

But output bits are linear functions of input bits!

XORTEA vs. TEA

"Hardware-friendlier" cipher,
since xor circuit is cheaper than add.

But output bits are linear functions of input bits!

e.g. First output bit is
$1 \oplus k_0 \oplus k_1 \oplus k_3 \oplus k_{10} \oplus k_{11} \oplus k_{12} \oplus k_{20} \oplus k_{21} \oplus k_{30} \oplus k_{32} \oplus$
$k_{33} \oplus k_{35} \oplus k_{42} \oplus k_{43} \oplus k_{44} \oplus k_{52} \oplus k_{53} \oplus k_{62} \oplus k_{64} \oplus k_{67} \oplus$
$k_{69} \oplus k_{76} \oplus k_{85} \oplus k_{94} \oplus k_{96} \oplus k_{99} \oplus k_{101} \oplus k_{108} \oplus k_{117} \oplus$
$k_{126} \oplus b_1 \oplus b_3 \oplus b_{10} \oplus b_{12} \oplus b_{21} \oplus b_{30} \oplus b_{32} \oplus b_{33} \oplus b_{35} \oplus$
$b_{37} \oplus b_{39} \oplus b_{42} \oplus b_{43} \oplus b_{44} \oplus b_{47} \oplus b_{52} \oplus b_{53} \oplus b_{57} \oplus b_{62}$.

Turning linearity into an attack

There is a matrix M with coefficients in $\mathbb{F}_2$
such that $\mathrm{XORTEA}_k(b) = (1, k, b)M$ for all (k, b).

Turning linearity into an attack

There is a matrix M with coefficients in $\mathbb{F}_2$
such that $\mathrm{XORTEA}_k(b) = (1, k, b)M$ for all (k, b).

$\mathrm{XORTEA}_k(b_1) \oplus \mathrm{XORTEA}_k(b_2) = (0, 0, b_1 \oplus b_2)M$.

Turning linearity into an attack

There is a matrix M with coefficients in $\mathbb{F}_2$ such that $\mathrm{XORTEA}_k(b) = (1, k, b)M$ for all (k, b).

$\mathrm{XORTEA}_k(b_1) \oplus \mathrm{XORTEA}_k(b_2) = (0, 0, b_1 \oplus b_2)M$.

Very fast attack: if $b_4 = b_1 \oplus b_2 \oplus b_3$ then $\mathrm{XORTEA}_k(b_1) \oplus \mathrm{XORTEA}_k(b_2) = \mathrm{XORTEA}_k(b_3) \oplus \mathrm{XORTEA}_k(b_4)$.

Turning linearity into an attack

There is a matrix M with coefficients in $\mathbb{F}_2$
such that $\mathrm{XORTEA}_k(b) = (1, k, b)M$ for all (k, b).

$\mathrm{XORTEA}_k(b_1) \oplus \mathrm{XORTEA}_k(b_2) = (0, 0, b_1 \oplus b_2)M$.

Very fast attack: if $b_4 = b_1 \oplus b_2 \oplus b_3$ then
$\mathrm{XORTEA}_k(b_1) \oplus \mathrm{XORTEA}_k(b_2) =$
$\mathrm{XORTEA}_k(b_3) \oplus \mathrm{XORTEA}_k(b_4)$.

This breaks PRP (and PRF):
uniform random permutation (or function) F
almost never has $F(b_1) \oplus F(b_2) = F(b_3) \oplus F(b_4)$.

TEA again

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

LEFTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y<<5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x<<5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Breaking LEFTEA

Addition is not $\mathbb{F}_2$-linear,
but addition mod 2 is $\mathbb{F}_2$-linear.

First output bit is $1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.

Breaking LEFTEA

Addition is not $\mathbb{F}_2$-linear,
but addition mod 2 is $\mathbb{F}_2$-linear.

First output bit is $1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.

Higher output bits are increasingly nonlinear
but they never affect first bit.

Breaking LEFTEA

Addition is not $\mathbb{F}_2$-linear,
but addition mod 2 is $\mathbb{F}_2$-linear.

First output bit is $1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.

Higher output bits are increasingly nonlinear
but they never affect first bit.

In TEA, >>5 **diffuses** nonlinear changes
from high bits to low bits. LEFTEA misses this.

Breaking LEFTEA

Addition is not $\mathbb{F}_2$-linear,
but addition mod 2 is $\mathbb{F}_2$-linear.

First output bit is $1 \oplus k_0 \oplus k_{32} \oplus k_{64} \oplus k_{96} \oplus b_{32}$.

Higher output bits are increasingly nonlinear
but they never affect first bit.

In TEA, >>5 **diffuses** nonlinear changes
from high bits to low bits. LEFTEA misses this.

(LEFTEA, like TEA, diffuses changes from low bits
to high bits via carries in addition and via <<4.)

TEA again

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

TEA4: another bad cipher

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 4;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Breaking TEA4

Fast attack:
$\mathrm{TEA4}_k(x + 2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same first bit.

Breaking TEA4

Fast attack:
$\mathrm{TEA4}_k(x + 2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same first bit.

(Why? Trace x, y differences through computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.)

Breaking TEA4

Fast attack:
$\mathrm{TEA4}_k(x + 2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same first bit.

(Why? Trace x, y differences through computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.)

Uniform random function F: probability $1/2$ that $F(x + 2^{31}, y)$ and $F(x, y)$ have same first bit.

Breaking TEA4

Fast attack:
$\mathrm{TEA4}_k(x + 2^{31}, y)$, $\mathrm{TEA4}_k(x, y)$ have same first bit.

(Why? Trace x, y differences through computation.
$r = 0$: multiples of $2^{31}, 2^{26}$.
$r = 1$: multiples of $2^{21}, 2^{16}$.
$r = 2$: multiples of $2^{11}, 2^{6}$.
$r = 3$: multiples of $2^{1}, 2^{0}$.)

Uniform random function F: probability $1/2$ that $F(x + 2^{31}, y)$ and $F(x, y)$ have same first bit.

This attack has PRF advantage $1/2$.

Differential attacks, linear attacks, etc.

More sophisticated attacks:
trace *probabilities* of differences;
probabilities of linear equations;
probabilities of higher-order differences
$C(x+\delta+\epsilon)-C(x+\delta)-C(x+\epsilon)+C(x)$; etc.
Use algebra and statistics to exploit
non-randomness in probabilities.

Differential attacks, linear attacks, etc.

More sophisticated attacks:
trace *probabilities* of differences;
probabilities of linear equations;
probabilities of higher-order differences
$C(x+\delta+\epsilon)-C(x+\delta)-C(x+\epsilon)+C(x)$; etc.
Use algebra and statistics to exploit
non-randomness in probabilities.

For TEA, these attacks get beyond $r=4$
but rapidly lose effectiveness as r increases.

Differential attacks, linear attacks, etc.

More sophisticated attacks:
trace *probabilities* of differences;
probabilities of linear equations;
probabilities of higher-order differences
$C(x+\delta+\epsilon)-C(x+\delta)-C(x+\epsilon)+C(x)$; etc.
Use algebra and statistics to exploit
non-randomness in probabilities.

For TEA, these attacks get beyond $r=4$
but rapidly lose effectiveness as r increases.

Hard question in cipher design:
How many "rounds" are really needed for security?

TEA again

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

REPTEA: another bad cipher

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0x9e3779b9;
  for (r = 0;r < 1000;r += 1) {

    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

Breaking REPTEA

$\mathrm{REPTEA}_k(b) = R_k^{1000}(b)$
where R_k does `x+=...;y+=...`.

Breaking REPTEA

$\mathrm{REPTEA}_k(b) = R_k^{1000}(b)$
where R_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.

Breaking REPTEA

$\mathrm{REPTEA}_k(b) = R_k^{1000}(b)$
where R_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.
Good chance that some b in list also has $a = R_k(b)$ in list. Then $\mathrm{REPTEA}_k(a) = R_k(\mathrm{REPTEA}_k(b))$.

Breaking REPTEA

$\mathrm{REPTEA}_k(b) = R_k^{1000}(b)$
where R_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.
Good chance that some b in list also has $a = R_k(b)$ in list. Then $\mathrm{REPTEA}_k(a) = R_k(\mathrm{REPTEA}_k(b))$.

For each (b, a) from list: Try solving equations $a = R_k(b)$ and $\mathrm{REPTEA}_k(a) = R_k(\mathrm{REPTEA}_k(b))$ to figure out k. (Exercise: Obtain more equations.)

Breaking REPTEA

$\mathrm{REPTEA}_k(b) = R_k^{1000}(b)$
where R_k does `x+=...;y+=...`.

Try list of 2^{32} inputs b.
Collect outputs $\mathrm{REPTEA}_k(b)$.
Good chance that some b in list also has $a = R_k(b)$ in list. Then $\mathrm{REPTEA}_k(a) = R_k(\mathrm{REPTEA}_k(b))$.

For each (b, a) from list: Try solving equations $a = R_k(b)$ and $\mathrm{REPTEA}_k(a) = R_k(\mathrm{REPTEA}_k(b))$ to figure out k. (Exercise: Obtain more equations.)

This is an example of a **slide attack.**
TEA avoids this by varying c.

Is original TEA secure?

```
void encrypt(uint32 *b,uint32 *k) {
  uint32 x = b[0], y = b[1];
  uint32 r, c = 0;
  for (r = 0;r < 32;r += 1) {
    c += 0x9e3779b9;
    x += y+c ^ (y<<4)+k[0] ^ (y>>5)+k[1];
    y += x+c ^ (x<<4)+k[2] ^ (x>>5)+k[3];
  }
  b[0] = x; b[1] = y;
}
```

A related-key property

Related keys: e.g., $\mathrm{TEA}_{k'}(b) = \mathrm{TEA}_k(b)$ where $(k'[0], k'[1], k'[2], k'[3]) =$
$(k[0] + 2^{31}, k[1] + 2^{31}, k[2], k[3])$.

A related-key property

Related keys: e.g., $\mathrm{TEA}_{k'}(b) = \mathrm{TEA}_k(b)$ where $(k'[0], k'[1], k'[2], k'[3]) = (k[0] + 2^{31}, k[1] + 2^{31}, k[2], k[3])$.

Is this an attack?

A related-key property

Related keys: e.g., $\mathrm{TEA}_{k'}(b) = \mathrm{TEA}_k(b)$ where $(k'[0], k'[1], k'[2], k'[3]) = (k[0] + 2^{31}, k[1] + 2^{31}, k[2], k[3])$.

Is this an attack?

PRP attack goal: distinguish TEA_k, for one secret key k, from uniform random permutation.

A related-key property

Related keys: e.g., $\mathrm{TEA}_{k'}(b) = \mathrm{TEA}_k(b)$ where $(k'[0], k'[1], k'[2], k'[3]) = (k[0] + 2^{31}, k[1] + 2^{31}, k[2], k[3])$.

Is this an attack?

PRP attack goal: distinguish TEA_k, for one secret key k, from uniform random permutation.

Brute-force attack: Guess key g;
see if TEA_g matches TEA_k on some outputs.

A related-key property

Related keys: e.g., $\mathrm{TEA}_{k'}(b) = \mathrm{TEA}_k(b)$ where $(k'[0], k'[1], k'[2], k'[3]) = (k[0] + 2^{31}, k[1] + 2^{31}, k[2], k[3])$.

Is this an attack?

PRP attack goal: distinguish TEA_k, for one secret key k, from uniform random permutation.

Brute-force attack: Guess key g;
see if TEA_g matches TEA_k on some outputs.

Related keys $\Rightarrow$ g succeeds with chance 2^{-126}.
Still very small.

More related-key properties

1997 Kelsey–Schneier–Wagner:
Fancier relationship between k, k' has chance 2^{-11}
of producing a particular output equation.

More related-key properties

1997 Kelsey–Schneier–Wagner:
Fancier relationship between k, k' has chance 2^{-11} of producing a particular output equation.

No evidence in literature that this helps brute-force attack, or otherwise affects PRP security. No challenge to security analysis of modes using TEA.

More related-key properties

1997 Kelsey–Schneier–Wagner:
Fancier relationship between k, k' has chance 2^{-11} of producing a particular output equation.

No evidence in literature that this helps brute-force attack, or otherwise affects PRP security. No challenge to security analysis of modes using TEA.

But advertised as "related-key cryptanalysis" and claimed to justify recommendations for designers regarding key scheduling.

Further reading

Some ways to learn more about cipher attacks, hash-function attacks, etc.:

- Read attack papers. Some useful sources:
  eSTREAM competition (2004–2008);
  SHA-3 competition (2007–2012);
  CAESAR competition (2013–2019);
  NISTLWC competition (2019–2023);
  FSE conference.
- Try to break ciphers yourself:
  e.g., find attacks on FEAL or on RC4.
  Reasonable starting point: 2000 Schneier
  "Self-study course in block-cipher cryptanalysis".

Bonus slide: reported ChaCha attack costs