(Symmetric) Cryptanalysis in Practice

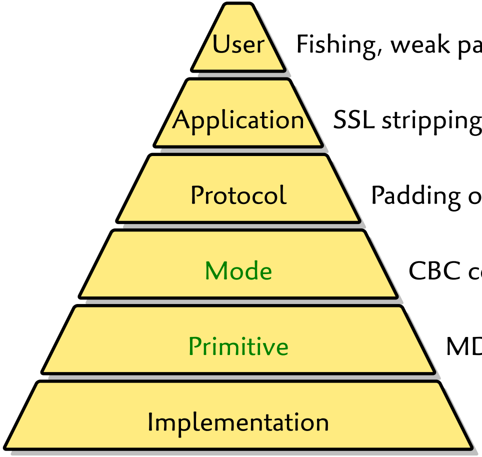
Gaëtan Leurent

Inria Paris, EPI COSMIQ

Cyber in Nancy
July 5, 2022

Cryptography and security

- Cryptography is an element to build a secure system
- There can be security issues at every step



User	Fishing, weak passwords, password reuse, ...
Application	SSL stripping, bad cert checks, encryption only, ...
Protocol	Padding oracle, predictible IV (BEAST)
Mode	CBC collisions (Sweet32), CTR malleability (Tor)
Primitive	MD5 collisions, RC4 bias
Implementation	Side channel, buffer overflow

Secure Cryptography

- Security is defined as a mathematical property
  - Discrete Log Problem: given g^x, finding x should be hard
  - AES-128 is expected to be a PRP
  - Protocols are proven secure assuming the primitives are secure
- Cryptographers build algorithm (primitive / mode / protocol)
  - Specific security goal: authenticity, integrity, ...
  - Specific assumptions: limits on message size, security model, random IVs, independent keys, ...

Classical approach

- Security of the protocol
  - Security proofs assuming security of cryptographic operations
- Security of the modes (HMAC, CBC, ...)
  - Security proofs (assuming security of the primitive)
- Security of the primitives (AES, SHA-1, RSA, ...)
  - Studied with cryptanalysis

Cryptanalysis

> *Anybody can design a system that he himself cannot break* [Bruce Schneier]

- We need public cryptanalysis research
  - Evaluation by the community
- Goal: replace weak algorithms before attacks are practical
  - We know that some government agencies attack weak cryptography

Cryptanalysis of primitives

- Evaluate new proposals and widely used standards
- Only way to evaluate their security

What is an attack?

For cryptographers

- Define expected security
- Anything faster is an attack
    - *Eg.* faster than trying all keys

For users

- Define attacker means
- Anything doable is an attack
    - *Eg.* one year on a PC

Attacks only get better

AES-256 has a 256-bit key

- Related-key attack with 2^{100} ops.
- Not a practical threat

Blowfish-32 has a 32-bit key

- No attack faster than 2^{32}
- Key-search takes minutes

What is an attack?

For cryptographers

- Define expected security
- Anything faster is an attack
  - *Eg.* faster than trying all keys

For users

- Define attacker means
- Anything doable is an attack
  - *Eg.* one year on a PC

Attacks only get better

AES-256 has a 256-bit key

- Related-key attack with 2^{100} ops.
- Not a practical threat

Blowfish-32 has a 32-bit key

- No attack faster than 2^{32}
- Key-search takes minutes

What is an attack?

For cryptographers

- Define expected security
- Anything faster is an attack
  - *Eg.* faster than trying all keys

For users

- Define attacker means
- Anything doable is an attack
  - *Eg.* one year on a PC

Attacks only get better

For cryptographers

- Attack primitive
- If broken, stop using it
  - Proof hypothesis broken

For users

- Does it break real protocols?
- Migration is expensive

Cryptanalysis in theory and in practice

Cryptanalysis of MD5

1993 Compression function attack

2005 Collision attack → *2007* Exploitable in APOP

2007 Free-start collision attack → *2009* Exploitable for rogue CA

↪ *2013* Exploited by Flame

Cryptanalysis of RC4

2000 Biases in RC4 keystream → *2013* Exploitable in TLS

2001 Related-key attack on RC4 → *2002* Exploitable in WEP

This talk

- Practical cryptanalysis of primitives
- Leverage weakness of crypto algorithms to break protocols

Cryptanalysis in theory and in practice

Cryptanalysis of MD5

1993 Compression function attack

2005 Collision attack → *2007* Exploitable in APOP

2007 Free-start collision attack → *2009* Exploitable for rogue CA

↪ *2013* Exploited by Flame

Cryptanalysis of RC4

2000 Biases in RC4 keystream → *2013* Exploitable in TLS

2001 Related-key attack on RC4 → *2002* Exploitable in WEP

This talk

- Practical cryptanalysis of primitives
- Leverage weakness of crypto algorithms to break protocols

Outline

Introduction

CBC Security
  CBC Collision Attack
  Attack in Practice: SWEET32

SHA-1 Chosen-prefix Collisions
  Record Computation
  PGP/GPG Impersonation

GSM security
  A5/1 Cryptanalysis
  A5/2 Cryptanalysis

GPRS Encryption
  GEA-1 Cryptanalysis
  GEA-2 Cryptanalysis

Outline

CBC Security

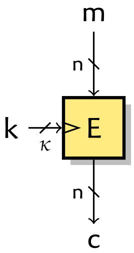
CBC Collision Attack

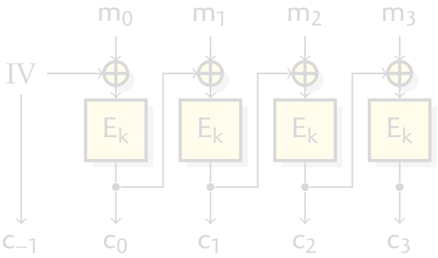
Attack in Practice: SWEET32

K. Bhargavan, G. L.
On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN
ACM CCS 2016,

Block ciphers and Modes of operation

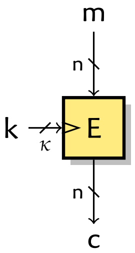
- A block cipher is a family of permutations
- It is used with a mode of operation: CBC, CTR, GCM, ...
  - To deal with variable-length messages
  - To include randomness
  - To reach various security goals (encryption, authentication, ...)
  - Important example: CBC: $c_i = E_k(m_i \oplus c_{i-1})$

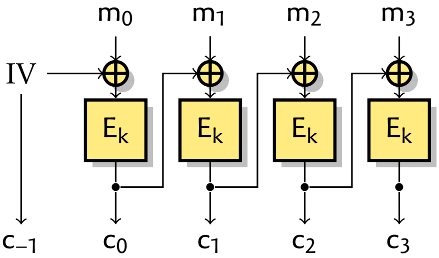




Block ciphers and Modes of operation

- A block cipher is a family of permutations
- It is used with a mode of operation: CBC, CTR, GCM, ...
  - To deal with variable-length messages
  - To include randomness
  - To reach various security goals (encryption, authentication, ...)
  - Important example: CBC: $c_i = E_k(m_i \oplus c_{i-1})$





Security of modes of operation

- Modes are proven secure assuming the block cipher is secure.
- Most modes (CBC, CTR, GCM, ...) have a security proof like:

$$\mathrm{Adv}^{\mathrm{CPA}}_{\mathrm{CBC\text{-}E}}(q,t) \leq \mathrm{Adv}^{\mathrm{PRP}}_{\mathrm{E}}(q',t') + \frac{\sigma^2}{2^n}$$

- The CPA security of CBC is essentially the PRP security of E (the block cipher)
- As long as the number of encrypted blocks $\sigma \lll 2^{n/2}$
  - Usually matching attack with birthday complexity ($2^{n/2}$)

Security of modes of operation

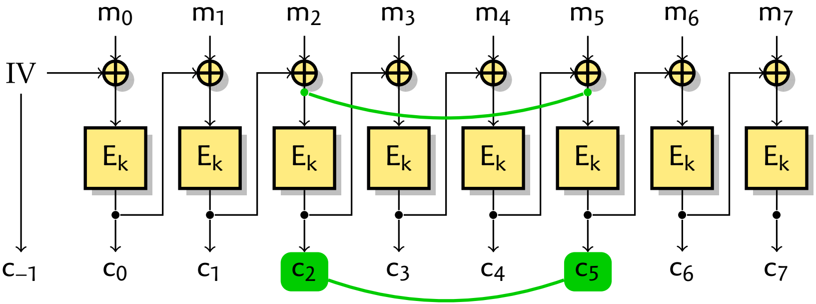
- Modes are proven secure assuming the block cipher is secure.
- Most modes (CBC, CTR, GCM, ...) have a security proof like:

$$\mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{CBC\text{-}E}}(q,t) \le \mathsf{Adv}^{\mathsf{PRP}}_{\mathsf{E}}(q',t') + \frac{\sigma^2}{2^n}$$

- The CPA security of CBC is essentially the PRP security of E (the block cipher)
- As long as the number of encrypted blocks $\sigma \lll 2^{n/2}$
  - Usually matching attack with birthday complexity ($2^{n/2}$)

CBC collisions

- Well known collision attack against CBC

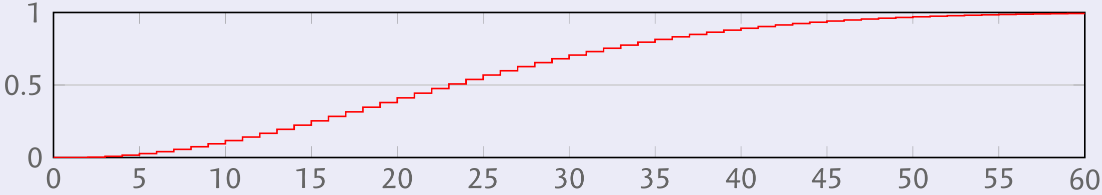


- If $c_i = c_j$, then $c_{i-1} \oplus m_i = c_{j-1} \oplus m_j$
- Ciphertext collision reveals the xor of two plaintext blocks

Birthday paradox

The birthday paradox

- In a room with 23 people, there is a 50% chance that two of them share the same birthday.



Security of CBC

- CBC leaks plaintext after $2^{n/2}$ blocks encrypted with the same key
- Security of mode can be lower than security of cipher

Birthday paradox

The birthday paradox

- Draw r random values from $[0, N-1]$
  - Constant probability of having a collision with $r = \Theta(\sqrt{N})$
  - Expected number of collisions is about $r^2/2N$
- Variant: Let $\mathcal{A}, \mathcal{B}$ be random subsets of $[0, N-1]$
  - $\mathcal{A} \cap \mathcal{B} \neq \emptyset$ with constant probability if $|\mathcal{A}| = |\mathcal{B}| = \sqrt{N}$
  - Expected number of matches $|\mathcal{A} \cap \mathcal{B}| \approx |\mathcal{A}| \times |\mathcal{B}|/N$

Security of CBC

- CBC leaks plaintext after $2^{n/2}$ blocks encrypted with the same key
- Security of mode can be lower than security of cipher

Communication issues

What cryptographers say [Rogaway 2011]

[Birthday] attacks can be a serious concern when employing a blockcipher of $n = 64$ bits, requiring relatively frequent rekeying to keep $\sigma \ll 2^{32}$

What standards say [ISO SC27 SD12]

The maximum amount of plaintext that can be encrypted before rekeying must take place is $2^{n/2}$ blocks, due to the birthday paradox.
As long as the implementation of a specific block cipher do not exceed these limits, using the block cipher will be safe.

What implementation did (in 2016)

TLS libraries, web browsers no rekeying

OpenVPN no rekeying (PSK mode) / rekey every hour (TLS mode)

Communication issues

What cryptographers say *[Rogaway 2011]*

[Birthday] attacks can be a serious concern when employing a blockcipher of $n = 64$ *bits, requiring relatively frequent rekeying to keep* $\sigma \ll 2^{32}$

What standards say *[ISO SC27 SD12]*

The maximum amount of plaintext that can be encrypted before rekeying must take place is $2^{n/2}$ *blocks, due to the birthday paradox.*
As long as the implementation of a specific block cipher do not exceed these limits, using the block cipher will be safe.

What implementation did (in 2016)

TLS libraries, web browsers no rekeying

OpenVPN no rekeying (PSK mode) / rekey every hour (TLS mode)

Communication issues

What cryptographers say [Rogaway 2011]

[Birthday] attacks can be a serious concern when employing a blockcipher of $n = 64$ *bits, requiring relatively frequent rekeying to keep* $\sigma \ll 2^{32}$

What standards say [ISO SC27 SD12]

The maximum amount of plaintext that can be encrypted before rekeying must take place is $2^{n/2}$ *blocks, due to the birthday paradox.*
As long as the implementation of a specific block cipher do not exceed these limits, using the block cipher will be safe.

What implementation did (in 2016)

TLS libraries, web browsers no rekeying

OpenVPN no rekeying (PSK mode) / rekey every hour (TLS mode)

Impact

- How bad is it?
  - Is it bad to leak a few xors of blocks of plaintexts?
  - Do applications encrypt enough data under the same key?
- 64-bit block cipher used in important protocols
  - 64-bit ciphers with CBC were the norm before AES
  - With a 64-bit block cipher, first collision around 32GB!
  - Blowfish-CBC in OpenVPN (default cipher in 2016)
  - 3DES-CBC in TLS (around 1-2% in 2016)
  - Kasumi in 3G (UMTS)
- Collision attacks usually not considered a practical threat
  - `openssl ciphers HIGH` used to be sorted by key length
    - Before 2014: `AES256, CAMELLIA256, 3DES, AES128, CAMELLIA128`
    - After 2014: `AES256, CAMELLIA256, AES128, CAMELLIA128, 3DES`

Impact

- How bad is it?
  - Is it bad to leak a few xors of blocks of plaintexts?
  - Do applications encrypt enough data under the same key?
- 64-bit block cipher used in important protocols
  - 64-bit ciphers with CBC were the norm before AES
  - With a 64-bit block cipher, first collision around 32GB!
  - Blowfish-CBC in OpenVPN (default cipher in 2016)
  - 3DES-CBC in TLS (around 1-2% in 2016)
  - Kasumi in 3G (UMTS)
- Collision attacks usually not considered a practical threat
  - `openssl ciphers HIGH` used to be sorted by key length
    - Before 2014: `AES256, CAMELLIA256, 3DES, AES128, CAMELLIA128`
    - After 2014: `AES256, CAMELLIA256, AES128, CAMELLIA128, 3DES`

Impact

- How bad is it?
  - Is it bad to leak a few xors of blocks of plaintexts?
  - Do applications encrypt enough data under the same key?

- 64-bit block cipher used in important protocols
  - 64-bit ciphers with CBC were the norm before AES
  - With a 64-bit block cipher, first collision around 32GB!
  - Blowfish-CBC in OpenVPN (default cipher in 2016)
  - 3DES-CBC in TLS (around 1-2% in 2016)
  - Kasumi in 3G (UMTS)

- Collision attacks usually not considered a practical threat
  - `openssl ciphers HIGH` used to be sorted by key length
    - Before 2014: `AES256, CAMELLIA256, 3DES, AES128, CAMELLIA128`
    - After 2014: `AES256, CAMELLIA256, AES128, CAMELLIA128, 3DES`

Towards a practical attack

- Assume a fixed message is repeatedly encrypted (under a fixed key)
  - Including a high value secret (cookie, password, ...) a few blocks
  - And some known/predictable sections (headers, ...) 2^t blocks
- Each collision reveals the xor of two plaintext blocks
- With some luck, xor of a known value and the secret

$$\underbrace{\texttt{cookie}}_{\text{unknown}} \oplus \underbrace{\texttt{header}}_{\text{known}} = \underbrace{c_{i-1} \oplus c_{j-1}}_{\text{known}}$$

  - Recover secret: $\texttt{cookie} = \texttt{header} \oplus c_{i-1} \oplus c_{j-1}$

- Concrete target: 3DES usage in HTTPS

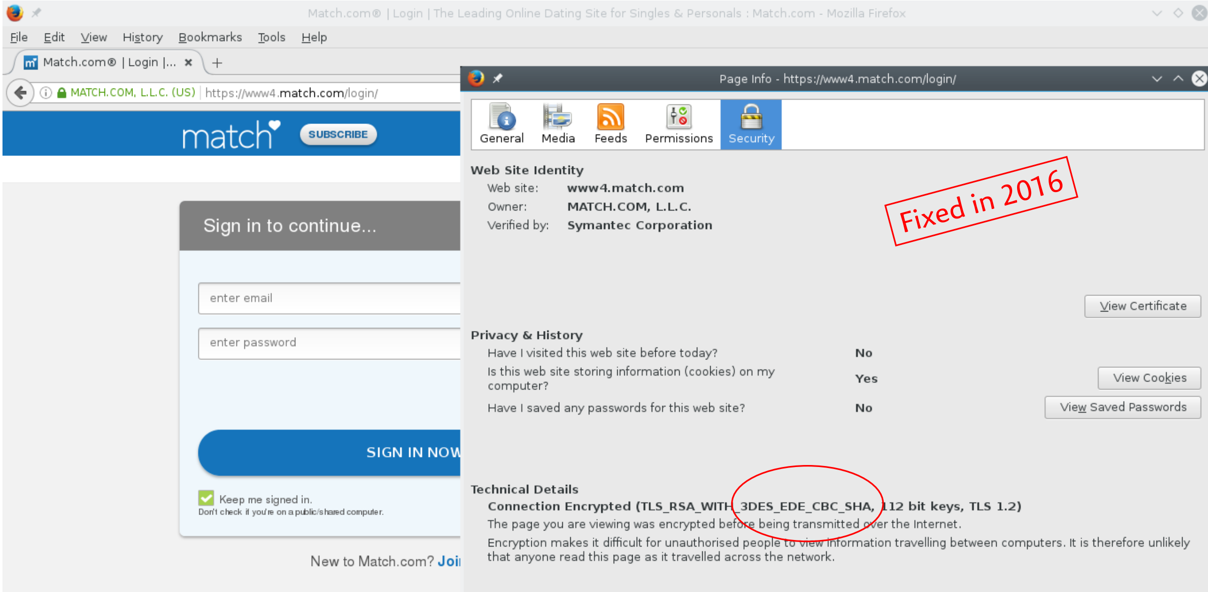
Towards a practical attack

- Assume a fixed message is repeatedly encrypted (under a fixed key)
  - Including a high value secret (cookie, password, ...) a few blocks
  - And some known/predictable sections (headers, ...) 2^t blocks
- Each collision reveals the xor of two plaintext blocks
- With some luck, xor of a known value and the secret

$$\underbrace{\texttt{cookie}}_{\text{unknown}} \oplus \underbrace{\texttt{header}}_{\text{known}} = \underbrace{c_{i-1} \oplus c_{j-1}}_{\text{known}}$$

  - Recover secret: $\texttt{cookie} = \texttt{header} \oplus c_{i-1} \oplus c_{j-1}$
- Concrete target: 3DES usage in HTTPS

Poorly configured websites



Web Site Identity

Web site: **www4.match.com**

Owner: **MATCH.COM, L.L.C.**

Verified by: **Symantec Corporation**

Privacy & History

Have I visited this web site before today? **No**

Is this web site storing information (cookies) on my computer? **Yes**

Have I saved any passwords for this web site? **No**

Technical Details

Connection Encrypted (TLS_RSA_WITH_3DES_EDE_CBC_SHA, 112 bit keys, TLS 1.2)

The page you are viewing was encrypted before being transmitted over the Internet.

Encryption makes it difficult for unauthorised people to view information travelling between computers. It is therefore unlikely that anyone read this page as it travelled across the network.

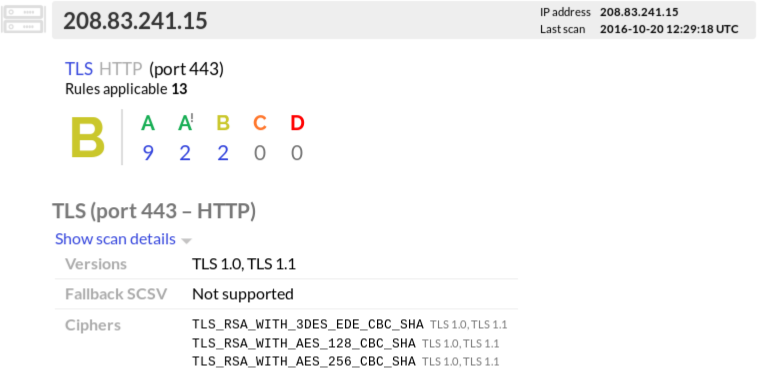
Fixed in 2016

Poorly configured websites

TLS cipher negotiation

- Client sends ordered list of supported ciphersuites
- Server chooses ciphersuite

https://discovery.cryptosense.com/analyze/208.83.241.15



208.83.241.15

IP address 208.83.241.15
Last scan 2016-10-20 12:29:18 UTC

TLS HTTP (port 443)
Rules applicable **13**

B

A	A!	B	C	D
9	2	2	0	0

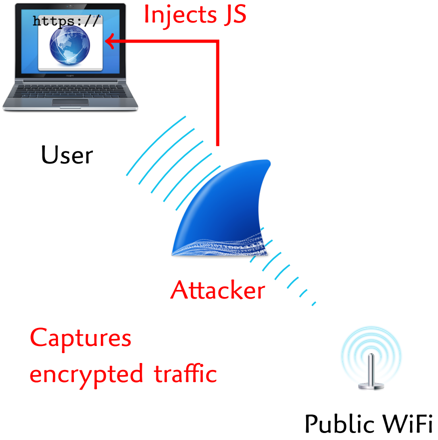
TLS (port 443 – HTTP)

Show scan details

Versions	TLS 1.0, TLS 1.1
Fallback SCSV	Not supported
Ciphers	TLS_RSA_WITH_3DES_EDE_CBC_SHA TLS 1.0, TLS 1.1 TLS_RSA_WITH_AES_128_CBC_SHA TLS 1.0, TLS 1.1 TLS_RSA_WITH_AES_256_CBC_SHA TLS 1.0, TLS 1.1

BEAST Attack Setting

[Duong & Rizzo 2011]



- Attacker has access to the network (*eg.* public WiFi)

1. Attacker uses JS to generate traffic
   - Tricks victim to malicious site
   - JS makes *cross-origin* requests
2. Attacker captures encrypted data

- Very powerful model
  Chosen plaintext

HTTP authentication tokens

- HTTP is stateless: authentication tokens sent with every request
- Also sent with *cross-origin* requests to allow “Facebook button”

HTTP Basic Auth (RFC 7617)

- User/Password sent in a header (base64 encoded)

```
Authorization: Basic dGVzdDoxMjPCow=
```

HTTP Cookies (RFC 6265)

1. User sends password in a from
2. Server reply with a Cookie
3. Cookie is included in every subsequent request

```
Cookie: C=123456
```

BEAST collision attack

- Assume user logged-in to secure website
- Javascript generates queries to HTTPS website
  - Including high value secret — a few blocks
  - And known content — 2^t blocks
- Each collision reveals the xor of two plaintext blocks
- Eventually a collision will reveal the secret
- Success after roughly 2^t collisions
  - $2^{n/2-t/2}$ queries, $2^{n/2+t/2}$ blocks
  - Tradeoff between # queries and total amount of data
- If rekeying after $2^{n/2}$ blocks, attack still possible
  - $2^{n/2}$ queries, $2^{n/2+t}$ blocks

worker.js

```
var url = "https://target";
var xhr = new XMLHttpRequest;

while(true) {
  xhr.open("HEAD", url, false);
  xhr.withCredentials = true;
  xhr.send();
  xhr.abort();
}
```

BEAST collision attack

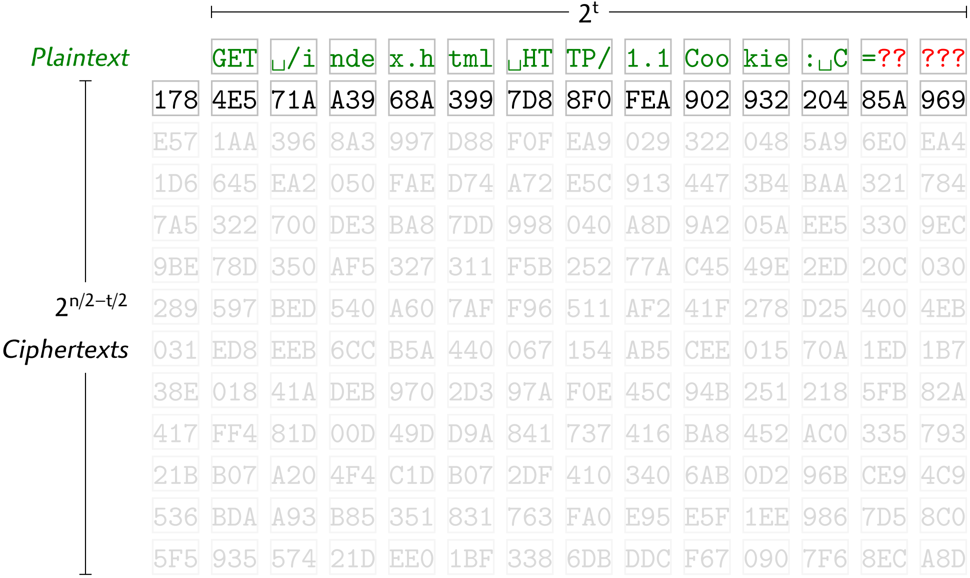
2^t

$2^{n/2-t/2}$

Ciphertexts

Plaintext	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



2^t

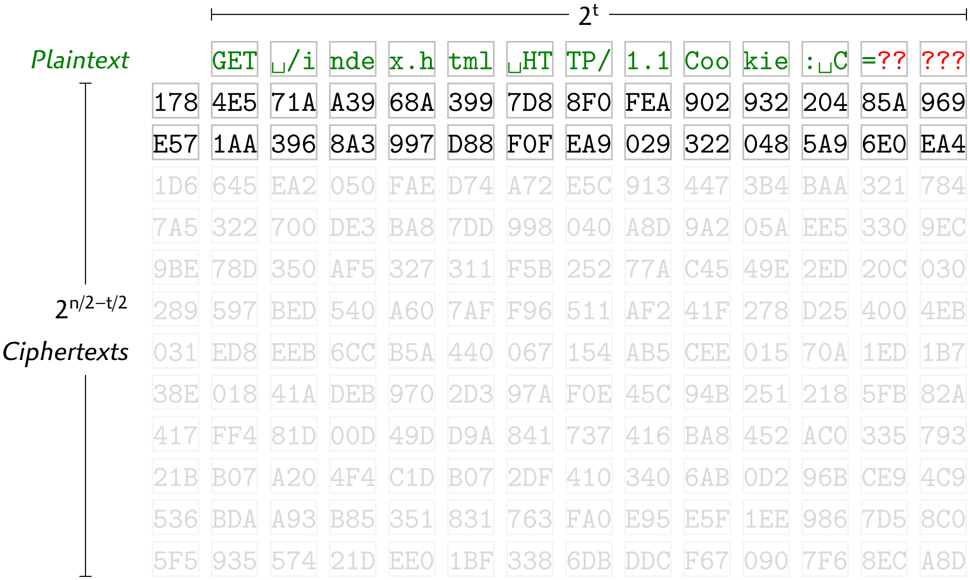
Plaintext

$2^{n/2-t/2}$

Ciphertexts

Plaintext		GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
Ciphertexts	178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
	E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
	1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
	7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
	9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
	289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
	031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
	38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
	417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
	21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
	536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
	5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

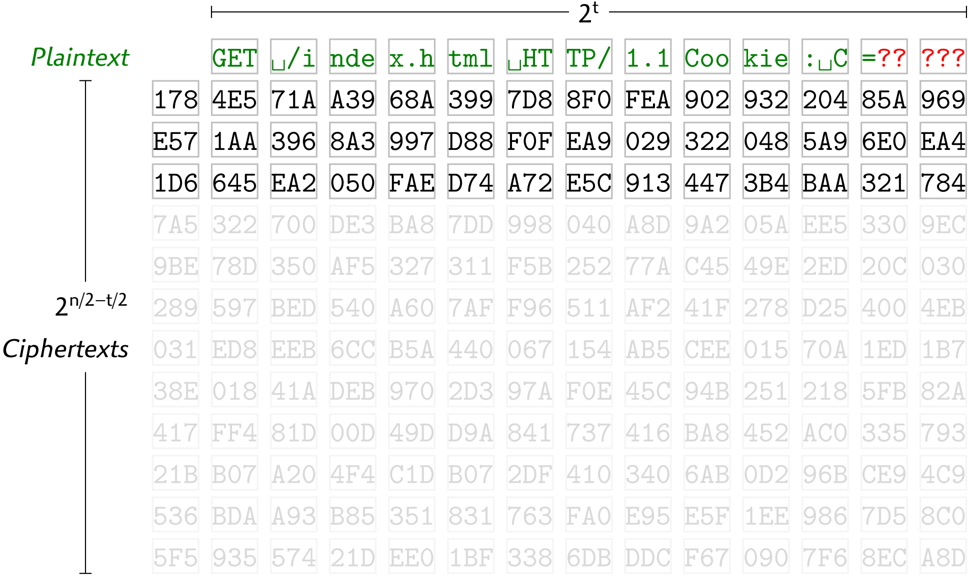
BEAST collision attack



2^t

Plaintext		GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
$2^{n/2-t/2}$ Ciphertexts	178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
	E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
	1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
	7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
	9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
	289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
	031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
	38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
	417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
	21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
	536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
	5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

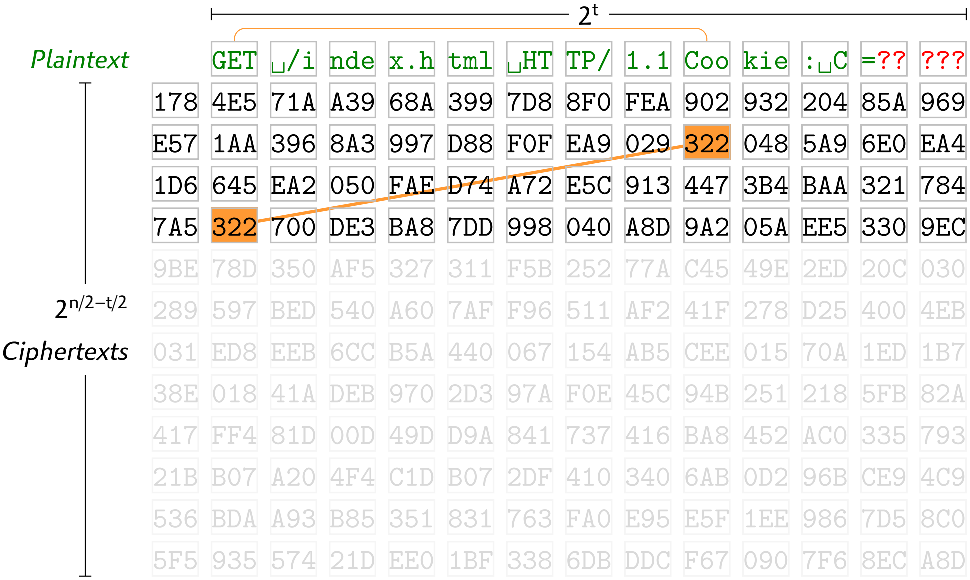
BEAST collision attack



2^t

Plaintext		GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
$2^{n/2-t/2}$ *Ciphertexts*	178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
	E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
	1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
	7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
	9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
	289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
	031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
	38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
	417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
	21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
	536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
	5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



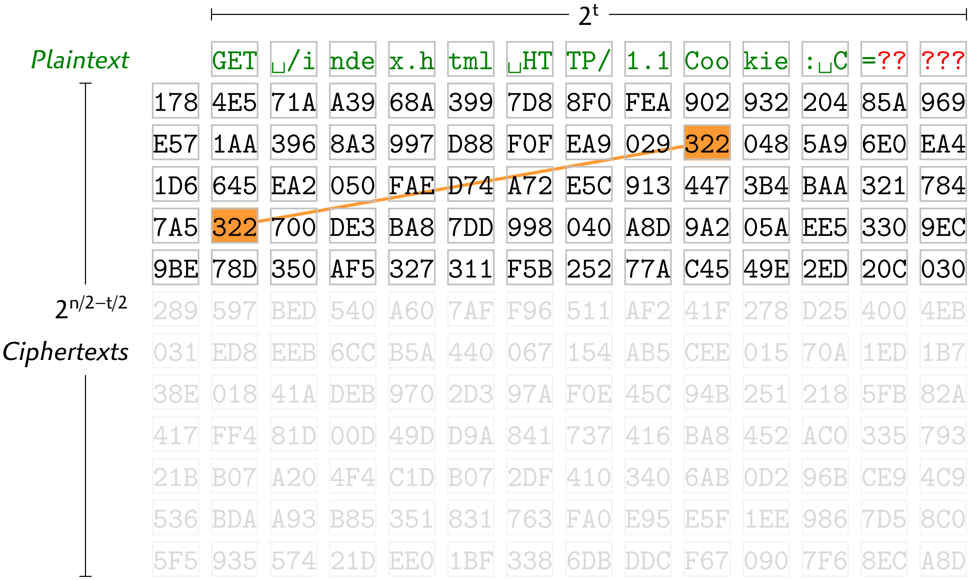
2^t

Plaintext		GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
	178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
	E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
	1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
	7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
	9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
	289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
	031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
	38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
	417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
	21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
	536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
	5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

$2^{n/2-t/2}$

Ciphertexts

BEAST collision attack



2^t

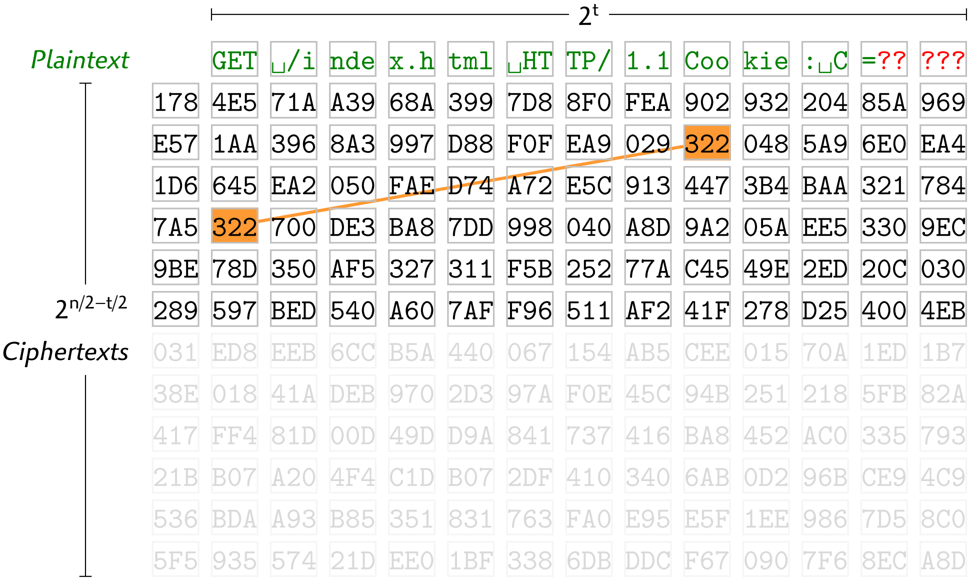
Plaintext

$2^{n/2-t/2}$

Ciphertexts

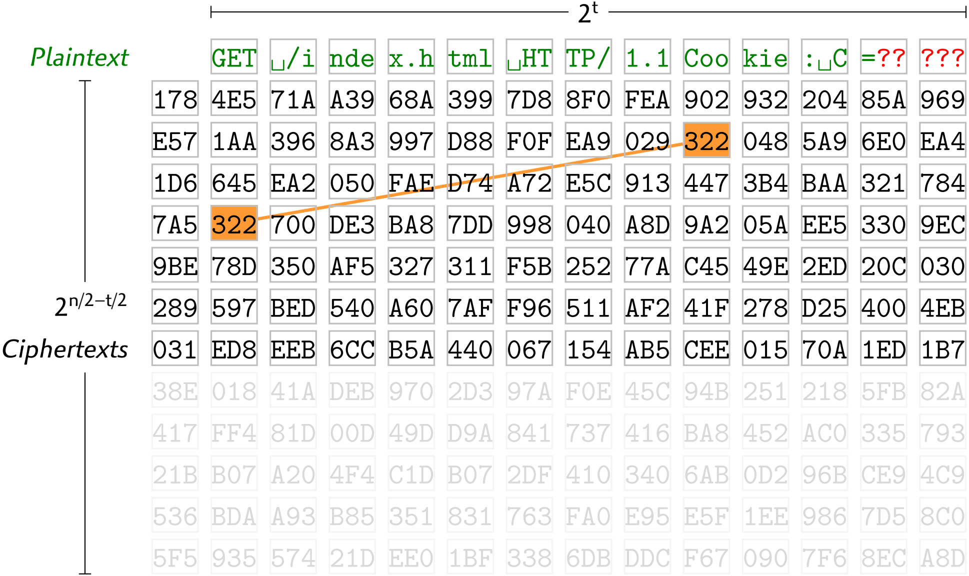
	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



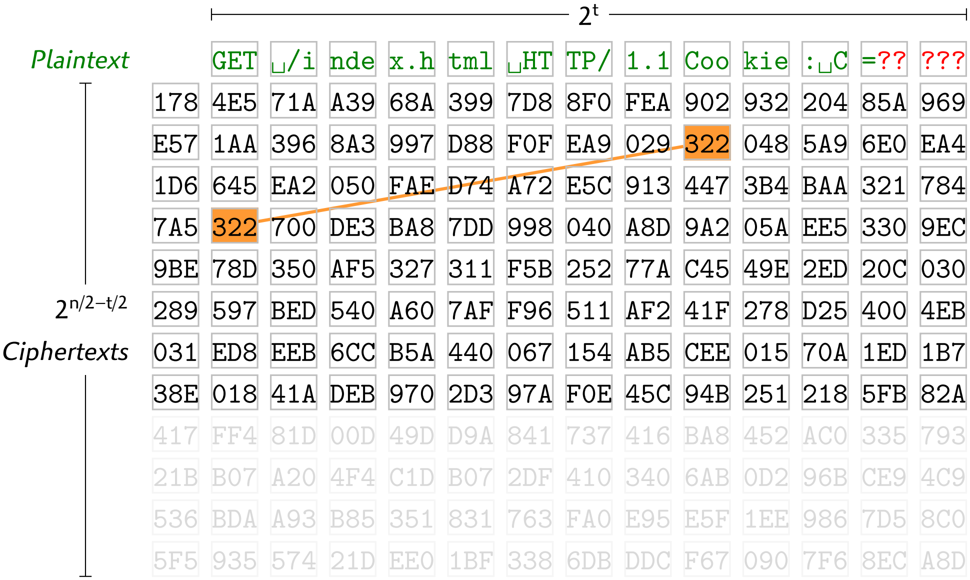
Plaintext		GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
	178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
	E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
	1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
	7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
	9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
$2^{n/2-t/2}$	289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
Ciphertexts	031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
	38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
	417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
	21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
	536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
	5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



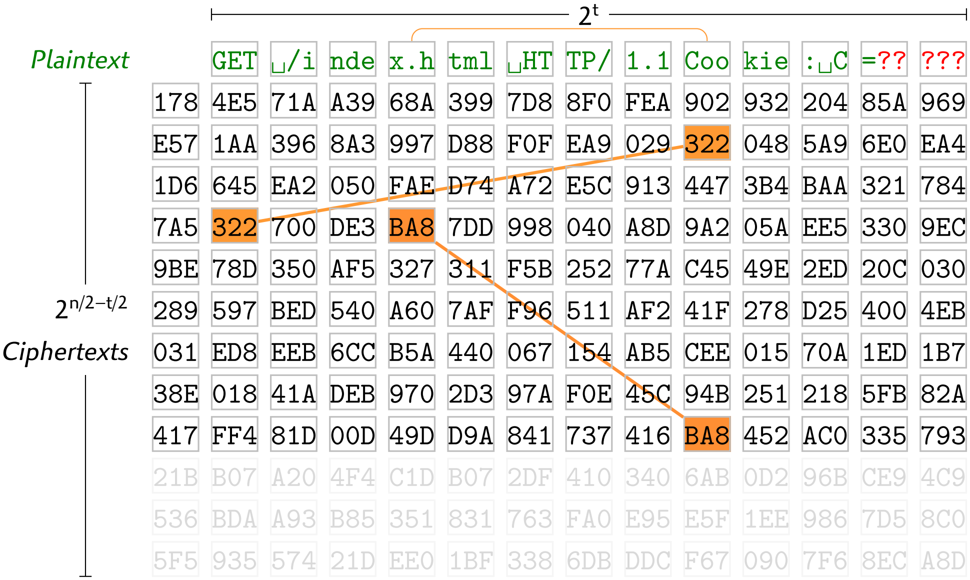
Plaintext		GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
	178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
	E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
	1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
	7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
	9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
$2^{n/2-t/2}$	289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
Ciphertexts	031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
	38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
	417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
	21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
	536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
	5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



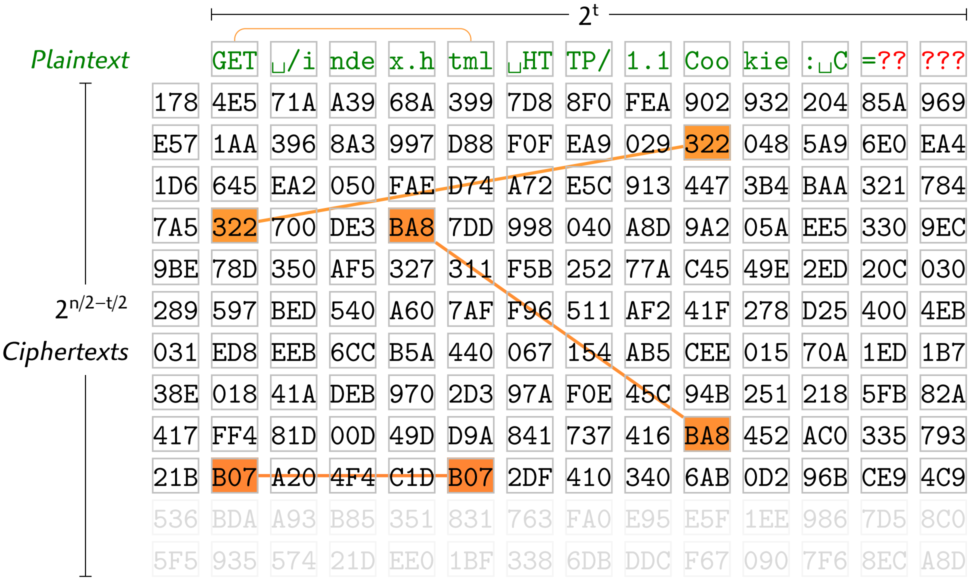
	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



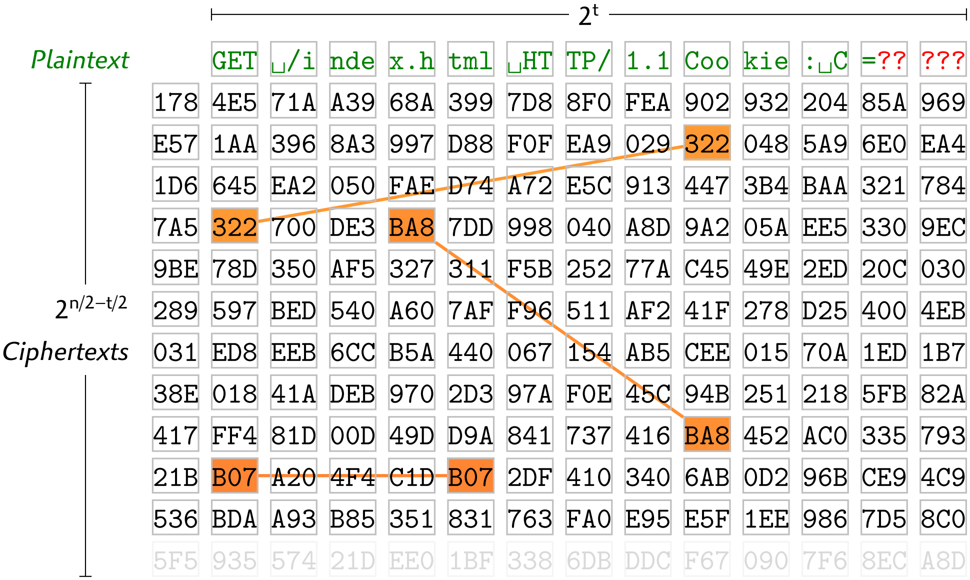
	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



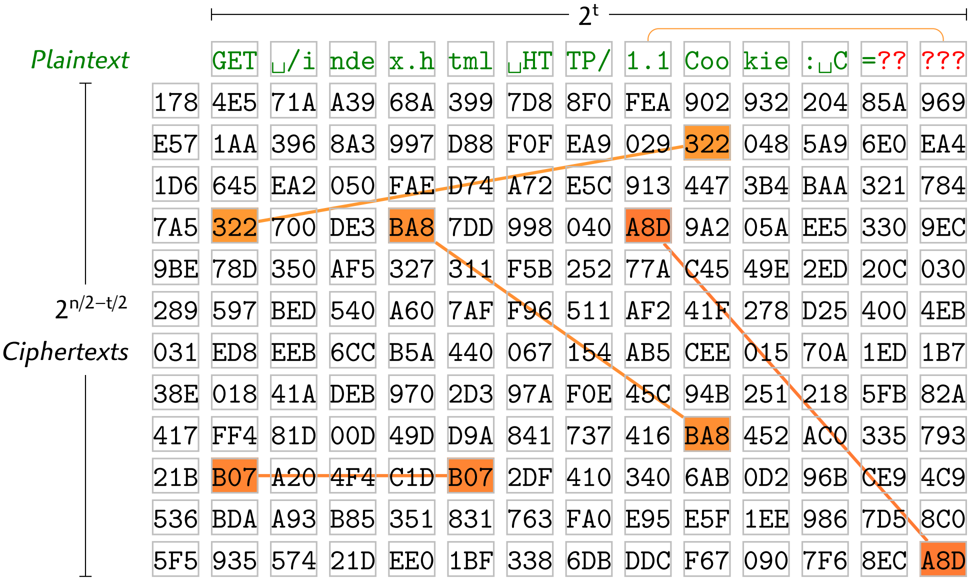
	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	322	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	322	700	DE3	BA8	7DD	998	040	A8D	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	BA8	452	AC0	335	793
21B	B07	A20	4F4	C1D	B07	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	A8D

BEAST collision attack



	GET	␣/i	nde	x.h	tml	␣HT	TP/	1.1	Coo	kie	:␣C	=??	???
178	4E5	71A	A39	68A	399	7D8	8F0	FEA	902	932	204	85A	969
E57	1AA	396	8A3	997	D88	F0F	EA9	029	**322**	048	5A9	6E0	EA4
1D6	645	EA2	050	FAE	D74	A72	E5C	913	447	3B4	BAA	321	784
7A5	**322**	700	DE3	**BA8**	7DD	998	040	**A8D**	9A2	05A	EE5	330	9EC
9BE	78D	350	AF5	327	311	F5B	252	77A	C45	49E	2ED	20C	030
289	597	BED	540	A60	7AF	F96	511	AF2	41F	278	D25	400	4EB
031	ED8	EEB	6CC	B5A	440	067	154	AB5	CEE	015	70A	1ED	1B7
38E	018	41A	DEB	970	2D3	97A	F0E	45C	94B	251	218	5FB	82A
417	FF4	81D	00D	49D	D9A	841	737	416	**BA8**	452	AC0	335	793
21B	**B07**	A20	4F4	C1D	**B07**	2DF	410	340	6AB	0D2	96B	CE9	4C9
536	BDA	A93	B85	351	831	763	FA0	E95	E5F	1EE	986	7D5	8C0
5F5	935	574	21D	EE0	1BF	338	6DB	DDC	F67	090	7F6	8EC	**A8D**

BEAST collision attack

- Assume user logged-in to secure website
- Javascript generates queries to HTTPS website
  - Including high value secret — a few blocks
  - And known content — 2^t blocks
- Each collision reveals the xor of two plaintext blocks
- Eventually a collision will reveal the secret
- Success after roughly 2^t collisions
  - $2^{n/2-t/2}$ queries, $2^{n/2+t/2}$ blocks
  - Tradeoff between # queries and total amount of data
- If rekeying after $2^{n/2}$ blocks, attack still possible
  - $2^{n/2}$ queries, $2^{n/2+t}$ blocks

worker.js

```
var url = "https://target";
var xhr = new XMLHttpRequest;

while(true) {
  xhr.open("HEAD", url, false);
  xhr.withCredentials = true;
  xhr.send();
  xhr.abort();
}
```

BEAST collision attack

- Assume user logged-in to secure website
- Javascript generates queries to HTTPS website
  - Including high value secret — a few blocks
  - And known content — 2^t blocks
- Each collision reveals the xor of two plaintext blocks
- Eventually a collision will reveal the secret
- Success after roughly 2^t collisions
  - $2^{n/2-t/2}$ queries, $2^{n/2+t/2}$ blocks
  - Tradeoff between # queries and total amount of data
- If rekeying after $2^{n/2}$ blocks, attack still possible
  - $2^{n/2}$ queries, $2^{n/2+t}$ blocks

worker.js

```
var url = "https://target";
var xhr = new XMLHttpRequest;

while(true) {
  xhr.open("HEAD", url, false);
  xhr.withCredentials = true;
  xhr.send();
  xhr.abort();
}
```

Proof-of-concept Attack Demo

- Demo with Firefox (Linux), and IIS 6.0 (Windows Server 2003)
  - Default configuration of IIS 6.0 does not support AES
- Each HTTP request encrypted in TLS record, with fixed key

1. Generate traffic with malicious JavaScript
2. Capture on the network with `tcpdump`
3. Remove header, extract ciphertext at fixed position
4. Sort ciphertext (`stdxxl`), look for collisions

- Expected time: 38 hours for 785 GB (tradeoff query size / # query).
- In practice: 30.5 hours for 610 GB.

Another target

OpenVPN used Blowfish-CBC by default

CBC Summary

Block size does matter

- Birthday attack against CBC with $2^{n/2}$ data
- Protocols from the 90's still use 64-bit ciphers
- Attacks with 2^{32} data are practical

- Sweet32 attack disclosed in August 2016

- OpenVPN 2.4 has cipher negotiation defaulting to AES
- Mozilla has implemented data limits (1M records) in Firefox 51 (January 2017)
- OpenSSL moved 3DES to `LOW` category
- NIST limits 3DES to 2^{20} blocks per key

- Firefox and Chrome disabled 3DES in 2021

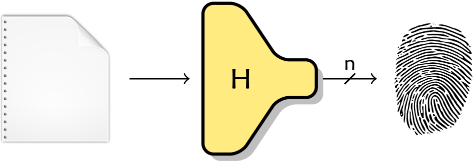
Outline

SHA-1 Chosen-prefix Collisions
    Record Computation
    PGP/GPG Impersonation

G. L., T. Peyrin
From Collisions to Chosen-Prefix Collisions — Application to Full SHA-1
Eurocrypt 2019

G. L., T. Peyrin
SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust
USENIX Security 2020

Hash functions



- Hash function: public function $\{0,1\}^* \to \{0,1\}^n$
  - Maps arbitrary-length message to fixed-length hash
- Hash function should behave like a random function
  - Hard to find collisions, preimages
  - Hash can be used as fingerprint, identifier
  - Used to instantiate the Random Oracle Model
- Used in many different contexts
  - Signature: hash-and-sign
  - MAC: hash-and-PRF, HMAC
  - Commitments, proof-of-work, ...

Concrete security goals

Preimage attack

Given F and $\overline{H}$, find M s.t. $F(M) = \overline{H}$. Ideal security: 2^n.

Second-preimage attack

Given F and M_1, find $M_2 \neq M_1$ s.t. $F(M_1) = F(M_2)$. Ideal security: 2^n.

Collision attack

Given F, find $M_1 \neq M_2$ s.t. $F(M_1) = F(M_2)$. Ideal security: $2^{n/2}$.

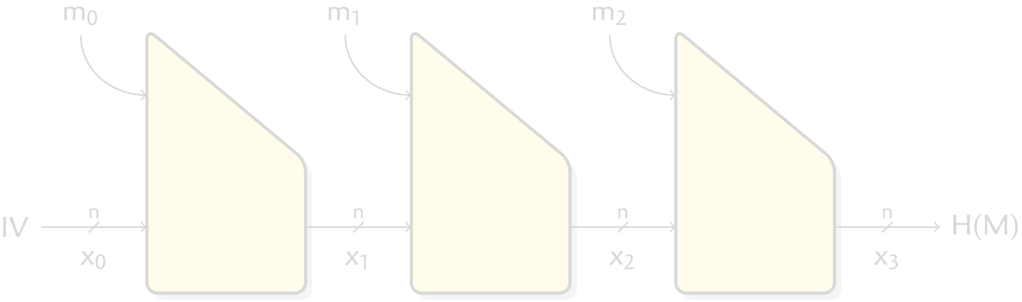
Collision search in practice

- Sort data to avoid quadratic complexity
- Pollard's rho (memoryless)
- Parallel collision search by van Oorschot and Wiener

SHA-1

- Designed by NSA: `SHA-0` [1993], then `SHA-1` [1995]
- Standardized by NIST, ISO, IETF, ...
- Widely used untill 2015

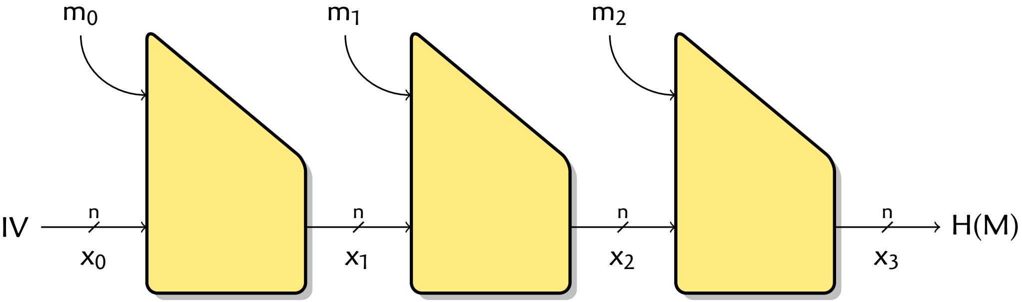
- Iterative structure: Merkle-Damgård construction ($n = 160$)
- Block cipher-based compression function: Davies-Meyer



SHA-1

- Designed by NSA: `SHA-0` [1993], then `SHA-1` [1995]
- Standardized by NIST, ISO, IETF, ...
- Widely used untill 2015

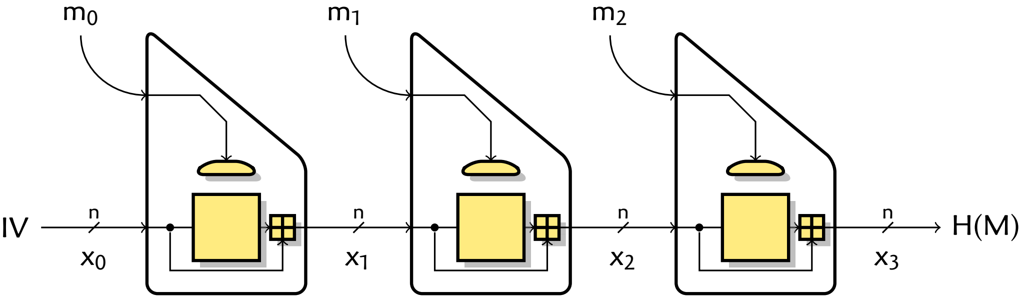
- Iterative structure: Merkle-Damgård construction ($n = 160$)
- Block cipher-based compression function: Davies-Meyer



SHA-1

- Designed by NSA: `SHA-0` [1993], then `SHA-1` [1995]
- Standardized by NIST, ISO, IETF, ...
- Widely used untill 2015

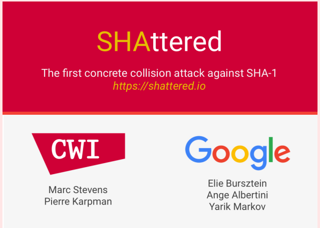
- Iterative structure: Merkle-Damgård construction ($n = 160$)
- Block cipher-based compression function: Davies-Meyer



SHA−1 Cryptanalysis

2005-02	Theoretical collision with 2^{69} op.	[Wang & al., Crypto'05]
…	Several unpublished collision attacks in the range $2^{51} - 2^{63}$	
2010-11	Theoretical collision with 2^{61} op.	[Stevens, EC'13]
2015-10	Practical freestart collision (on GPU)	[Stevens, Karpman & Peyrin, Eurocrypt'16]
2017-02	Practical collision with $2^{64.7}$ op. (GPU)	[Stevens & al., Crypto'17]

SHAttered attack: Colliding PDFs



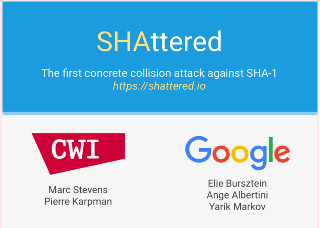
SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

CWI
Marc Stevens
Pierre Karpman

Google
Elie Bursztein
Ange Albertini
Yarik Markov

`SHA-1 =`
`38762cf7f55934b34d17`
`9ae6a4c80cadccbb7f0a`



SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

CWI
Marc Stevens
Pierre Karpman

Google
Elie Bursztein
Ange Albertini
Yarik Markov

SHA-1 Deprecation

2006-03 **NIST Policy on Hash Functions**
Federal agencies should stop using `SHA-1` for digital signatures, digital time stamping and other applications that require collision resistance as soon as practical, and must use the SHA-2 family of hash functions for these applications after 2010.

2011-11 **CA/Browser Forum**:
"`SHA-1` MAY be used until SHA-256 is supported widely by browsers"

2014-09 **CA/Browser Forum** depreciation plan

- Stop issuing `SHA-1` certificates on 2016-01-01
- Do not trust `SHA-1` certificates after 2017-01-01

2015-10 Browsers consider moving deadline to 2016-07

2017-0x **Modern browsers reject `SHA-1` certificates**

SHA−1 Usage in 2020

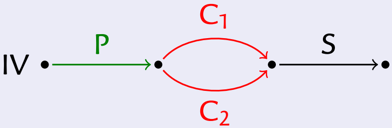
- `SHA−1` certificates (X.509) still exists
    - CAs sell legacy `SHA−1` certificates for legacy clients
    - Accepted by some non-web modern clients
- PGP signatures with `SHA−1` still trusted
    - Default hash for key certification in GnuPGv1 (legacy branch)
    - 1% of public certifications (Web-of-Trust) in 2019 used `SHA−1`
- `SHA−1` still allowed for in-protocol signatures in TLS, SSH
    - Used by 3% of Alexa top 1M servers
- DNSSEC supports and use `SHA−1` signatures
    - 18% of TLDs used `SHA−1` in 2020
- `HMAC−SHA−1` ciphersuites (TLS) are still used by 8% of Alexa top 1M servers
- Probably a lot of more obscure protocols...
    - EMV credit cards use weird `SHA−1` signatures

Chosen-Prefix Collisions [*Stevens, Lenstra & de Weger, EC'07*]

- Collisions are hard to exploit: garbage collision blocks C_i

Identical-prefix collision

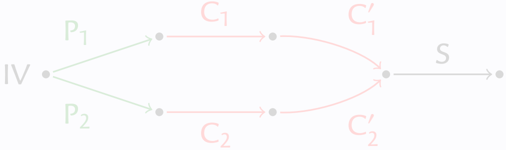
- Given IV, find $M_1 \neq M_2$ s. t. $H(M_1) = H(M_2)$



- Arbitrary common prefix/suffix, random collision blocks
- Breaks integrity verification
- Colliding PDFs (breaks signature?)

Chosen-prefix collision

- Given P_1, P_2, find $M_1 \neq M_2$ s. t. $H(P_1 \parallel M_1) = H(P_2 \parallel M_2)$



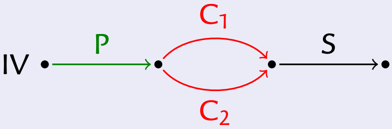
- Breaks certificates
  Rogue CA [Stevens & al, Crypto'09]
- Breaks TLS, SSH
  SLOTH [Bhargavan & L, NDSS'16]

Chosen-Prefix Collisions *[Stevens, Lenstra & de Weger, EC'07]*

- Collisions are hard to exploit: garbage collision blocks C_i

Identical-prefix collision

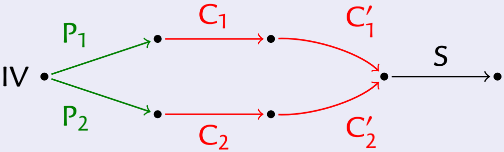
- Given IV, find $M_1 \neq M_2$ s. t. $H(M_1) = H(M_2)$



- Arbitrary common prefix/suffix, random collision blocks
- Breaks integrity verification
- Colliding PDFs (breaks signature?)

Chosen-prefix collision

- Given P_1, P_2, find $M_1 \neq M_2$ s. t. $H(P_1 \| M_1) = H(P_2 \| M_2)$



- Breaks certificates
  Rogue CA [Stevens & al, Crypto'09]
- Breaks TLS, SSH
  SLOTH [Bhargavan & L, NDSS'16]

Our results

Chosen-prefix collision attack on SHA-1

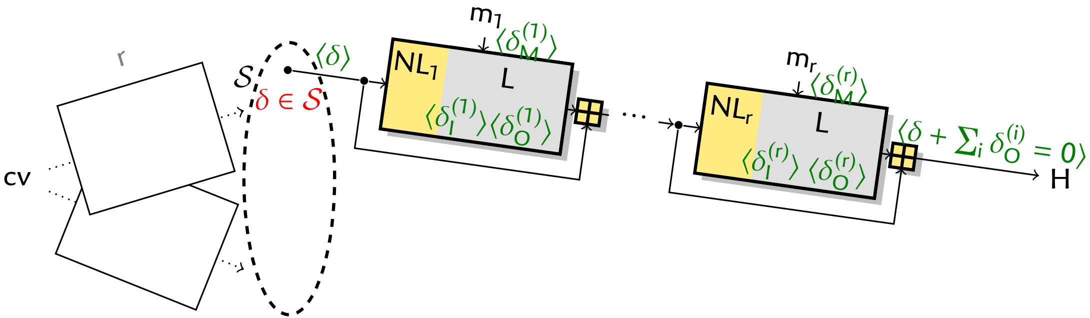
- Theoretical attack at Eurocrypt 2019 Complexity $2^{67.1}$
- Practical attack at USENIX 2020 Complexity $2^{63.4}$

1. Complexity improvements (factor 8 ∼ 10)

   identical-prefix collision from $2^{64.7}$ to $2^{61.2}$ (11 kUS$ in GPU rental)
   chosen-prefix collision from $2^{67.1}$ to $2^{63.4}$ (45 kUS$ in GPU rental)

2. Record computation
   - Implementation of the full CPC attack
   - 2 months using 900 GPU (GTX 1060)
3. PGP Web-of-Trust impersonation
   - 2 keys with different IDs and colliding certificates
   - Certification signature can be copied to the second key

Chosen-prefix collision attack on SHA-1 [L. & P., EC'19]



1. **Setup:** Find a set of "nice" chaining value differences $\mathcal{S}$
2. **Birthday phase:** Find m_1, m_1' such that $H(P_1 \| m_1) - H(P_2 \| m_1') \in \mathcal{S}$
3. **Near-collision phase:** Erase the state difference, using near-collision blocks

- Expected complexity $\approx 2^{64}$ [EC'19, USENIX'20]

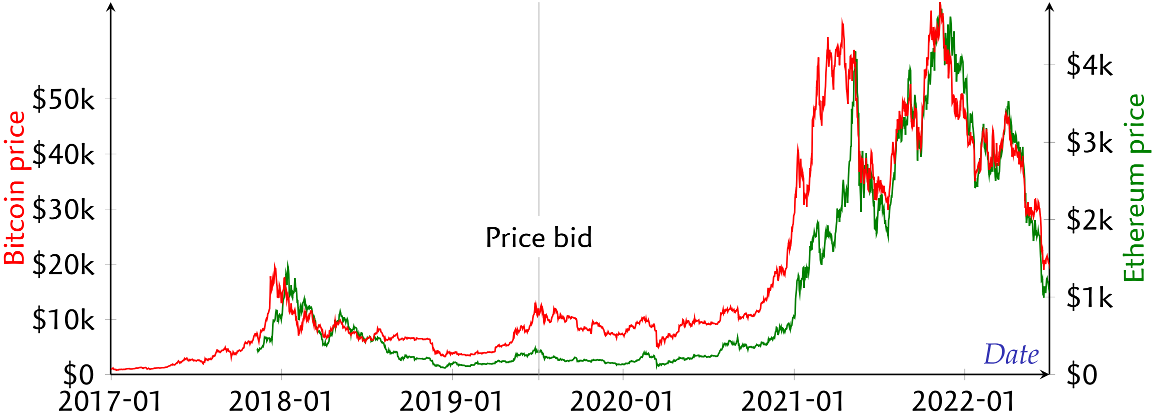
Running a 2^{64} computation on a budget

- Running the attack on Amazon/Google cloud GPU estimated to cost 160 kUS$ (spot/preemptible instances)
- After cryptocurrency crash in 2018, cheap GPU farms to rent!
  - 👍 3–4 times cheaper
    45 kUS$ with public prices on `gpuserversrental.com` (early 2020)
  - 👎 Gaming or mining-grade GTX cards (rather than Tesla)
  - 👎 Low-end CPUs
  - 👎 Slow internet link
  - 👎 No cluster management
  - 👎 Pay by month, not on-demand

- Pricing fluctuates together with cryptocurrencies prices

Running a 2^{64} computation on a budget

Bitcoin price history



- Pricing fluctuates together with cryptocurrencies prices

Birthday phase

Find m_1, m_2 such that $H(P_1 \| m_1) - H(P_2 \| m_2) \in \mathcal{S}$

- Set $\mathcal{S}$ of 2^{38} "nice" chaining value differences
- Birthday paradox: complexity about $\sqrt{2^{n+1}/|\mathcal{S}|} = 2^{61.5}$
- Chains of iterations to reduce the memory [van Oorschot & Wiener, CCS'94]
  - Truncate `SHA-1` to 96 bits, partial collision likely to be in $\mathcal{S}$
  - About 500GB of storage
  - Easy to parallelize on GPU
  - Expected complexity $\approx 2^{62}$, ($2^{26.4}$ truncated collisions)
- Success after one month
  - $2^{62.9}$ computations ($2^{27.7}$ truncated collisions)
  - Bad luck! ☹

Near-collision phase

Erase the state difference, using near-collision blocks

- Very technical part of the attack: each block similar to a collision attack
  - Find the useful output differences for the next block by exploring $\mathcal{S}$
  - Build a differential trail with specific input/output conditions
  - Build GPU code dedicated to the trail: neutral bits, boomerangs, ...
- For simplicity, we use variants of the trail of Stevens for all blocks
  - Reuse most neutral bits / boomerang analysis
  - Reuse most GPU code [Stevens, Bursztein, Karpman, Albertini & Markov, C'17]
- Aim for 10 blocks, expected complexity: $2^{62.8}$
  - Last block: $2^{61.6}$ (equivalent to collision attack)
  - Intermediate blocks: $2^{62.1}$ in total (each block is cheap)

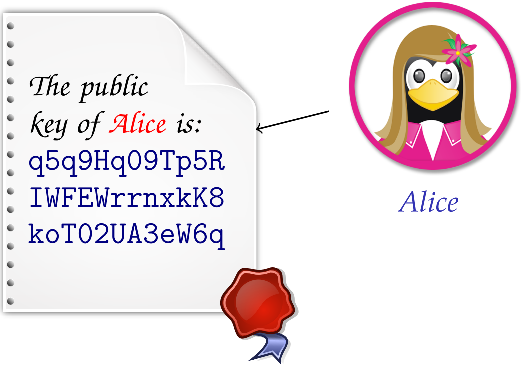
- Success after one month
  - 2^{62} computations (time lost when preparing the trails and GPU code)
  - Good luck! ☺

The First SHA-1 Chosen-prefix Collision

- 416-bit prefix
- 96 birthday bits
- 9 near-collision blocks

Message A	Message B
99040d047fe81780012000ff4b65792069732070617274206f66206120636f6c 6c6973696f6e212049742773206120747261702179c61af0afcc054515d9274e	99030d047fe81780011800ff50726163746963616c205348412d312063686f73 656e2d7072656669782063686f6c6c6973696f6e211d276c6ba661e1040e1f7d76
7307624b1dc7fb23988bb8de8b575dba7b9eab31c1674b6d974378a827732ff5 851c76a2e60772b5a47ce1eac40bb993c12d8c70e24a4f8d5fcdedc1b32c9cf1	7f076249ddc7fb332c8bb8c2b7575dbec79eab2be1674b7db34378b4cb732fe1 891c76a0260772a5107ce1f6e80bb9977d2d8c68524a4f9d5fcdedcd0b2c9ce1
9e31af2429759d42e4dfdb31719f587623ee552939b6dcdc459fca53553b70f8 7ede30a247ea3af6c759a2f20b320d760db64ff479084fd3ccb3cdd48362d96a	9231af26e9759d5250dfdb2d4d9f58729fee553319b6dccc619fca4fb93b70ec 72de30a087ea3ae67359a2ee27320d72b1b64fecc9084fc3ccb3cdd83b62d97a
9c430617caff6c36c637e53fde28417f626fec54ed7943a46e5f5730f2bb38fb 1df6e0090010d00e24ad78bf92641993608e8d158a789f34c46fe1e6027f35a4	904306150aff6c267237e523e228417bde6fec4ecd7943b44a5f572c1ebb38ef 11f6e00bc010d01e90ad78a3be641997dc8e8d0d3a789f24c46fe1eaba7f35b4
cbfb827076c50eca0e8b7cca69bb2c2b790259f9bf9570dd8d4437a3115faff7 c3cac09ad25266055c27104755178eaeff825a2caa2acfb5de64ce7641dc59a5	c7fb8272b6c50edaba8b7cd655bb2c2fc50259e39f9570cda94437bffd5fafe3 cfcac0981252661 5e827105b79178eaa43825a341a2acfa5de64ce7af9dc59b5
41a9fc9c756756e2e23dc713c8c24c9790aa6b0e38a7f55f14452a1ca2850ddd 9562fd9a18ad42496aa97008f74672f68ef461eb88b09933d626b4f918749cc0	4da9fc9eb56756f2563dc70ff4c24c932caa6b1418a7f54f30452a004e850dc9 9962fd98d8ad4259dea97014db4672f232f461f338b09923d626b4f5a0749cd0
27fddd6c425fc4216835d0134d15285bab2cb784a4f7cbb4fb514d4bf0f6237c f00a9e9f132b9a066e6fd17f6c42987478586ff651af96747fb426b9872b9a88	2bfddd6e825fc431dc35d00f7115285f172cb79e84f7cba4df514d571cf62368 fc0a9e9dd32b9a16da6fd16340429870c4586feee1af96647fb426b53f2b9a98
e4063f59bb334cc00650f83a80c42751b71974d300fc2819a2e8f1e32c1b51cb 18e6bfc4db9baef675d4aaf5b1574a047f8f6dd2ec153a93412293974d928f88	e8063f5b7b334cd0b250f826bcc427550b1974c920fc280986e8f1ffc01b51df 14e6bfc61b9baee6c1d4aae99d574a00c38f6dca5c153a834122939bf5928f98
ced9363cfef97ce2e742bf34c96b8ef3875676fea5cca8e5f7dea0bab2413d4d e00ee71ee01f162bdb6d1eafd925e6aebaae6a354ef17cf205a404fbdb12fc45	c2d9363e3ef97cf25342bf28f56b8ef73b5676e485cca8f5d3dea0a65e413d59 ec0ee71c201f163b6f6d1eb3f525e6aa06ae6a2dfef17ce205a404f76312fc55
4d41fdd95cf2459664a2ad032d1da60a73264075d7f1e0d6c1403ae7a0d861df 3fe5707188dd5e07d1589b9f8b6630553f8fc352b3e0c27da80bddba4c64020d	4141fddb9cf24586d0a2ad1f111da60ecf26406ff7f1e0c6e5403afb4cd861cb 33e5707348dd5e1765589b83a7663051838fc34a03e0c26da80bddb6f464021d

Attacking key certification [*Stevens, Lenstra & de Weger, EC'07*]


The public key of Alice is:
`q5q9Hq09Tp5R`
`IWFEWrrnxkK8`
`koT02UA3eW6q`

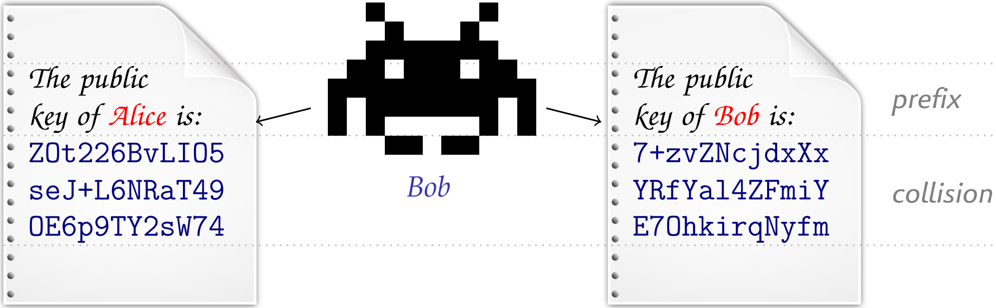
PKI Infrastructure

- Alice generates key
- Asks CA to sign
- Certificate proves ID

Impersonation attack

1. Bob creates keys s.t. $H(\texttt{Alice}\|k_A) = H(\texttt{Bob}\|k_B)$
2. Bob asks CA to certify his key k_B
3. Bob copies the signature to k_A, impersonates Alice

Attacking key certification [Stevens, Lenstra & de Weger, EC'07]



The public key of Alice is: Z0t226BvLI05 seJ+L6NRaT49 0E6p9TY2sW74

The public key of Bob is: 7+zvZNcjdxXx YRfYal4ZFmiY E70hkirqNyfm

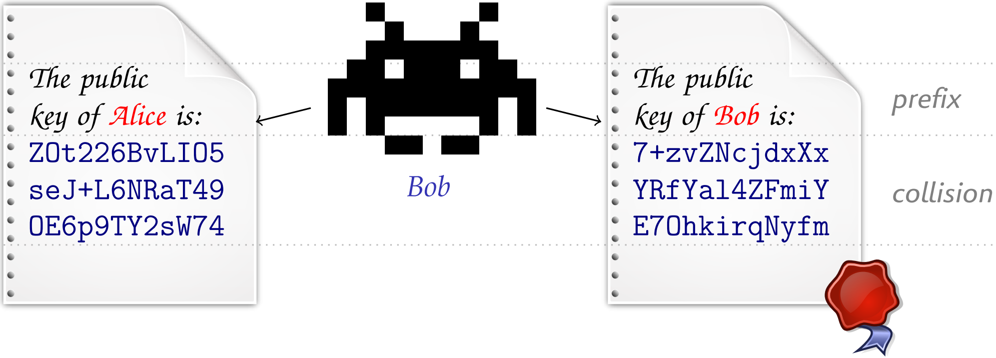
PKI Infrastructure

- Alice generates key
- Asks CA to sign
- Certificate proves ID

Impersonation attack

1. Bob creates keys s.t. $H(\texttt{Alice} \| k_A) = H(\texttt{Bob} \| k_B)$
2. Bob asks CA to certify his key k_B
3. Bob copies the signature to k_A, impersonates Alice

Attacking key certification [Stevens, Lenstra & de Weger, EC'07]



The public key of Alice is:
ZOt226BvLIO5
seJ+L6NRaT49
OE6p9TY2sW74

The public key of Bob is:
7+zvZNcjdxXx
YRfYal4ZFmiY
E7OhkirqNyfm

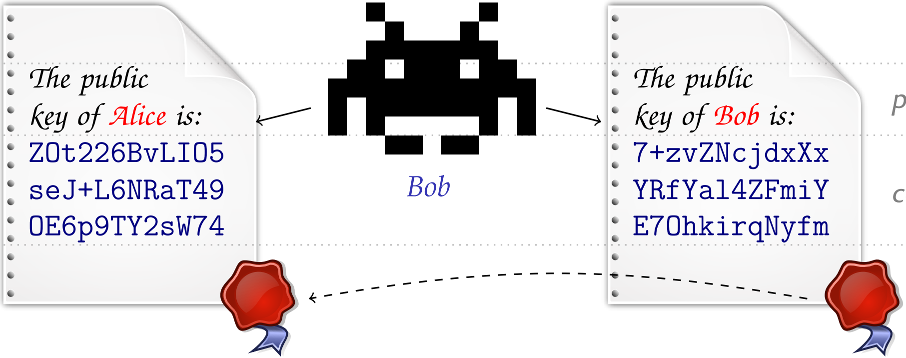
PKI Infrastructure

- Alice generates key
- Asks CA to sign
- Certificate proves ID

Impersonation attack

1. Bob creates keys s.t. $H(\texttt{Alice}\|k_A) = H(\texttt{Bob}\|k_B)$
2. Bob asks CA to certify his key k_B
3. Bob copies the signature to k_A, impersonates Alice

Attacking key certification [*Stevens, Lenstra & de Weger, EC'07*]



The public
key of Alice is:
`Z0t226BvLI05`
`seJ+L6NRaT49`
`0E6p9TY2sW74`

Bob

The public
key of Bob is:
`7+zvZNcjdxXx`
`YRfYal4ZFmiY`
`E70hkirqNyfm`

prefix

collision

PKI Infrastructure

- Alice generates key
- Asks CA to sign
- Certificate proves ID

Impersonation attack

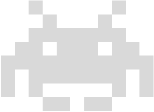
1. Bob creates keys s.t. $H(\texttt{Alice}\|k_A) = H(\texttt{Bob}\|k_B)$
2. Bob asks CA to certify his key k_B
3. Bob copies the signature to k_A, impersonates Alice

PGP identity certificates

- PGP identity certificate has public key first, UserID next
    - Each blob prefixed by length
    - Cannot just use the ID a prefix as with X.509 certificates
    - Quite rigid format (weird extensions not signed)

- Use keys of different length, fields misaligned
- PGP format supports for JPEG picture in key, and picture can be signed
    - JPEG readers ignore garbage after End of Image marker

- Certificate A has RSA-8192 public key, with victim ID
- Certificate B has RSA-6144 public key, and attacker's picture
    - Stuff JPEG in key A, and ID B in JPEG
    - Need very small JPEG: example 181-byte JPEG (*almost compliant*)

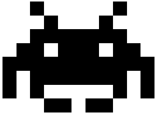


PGP identity certificates

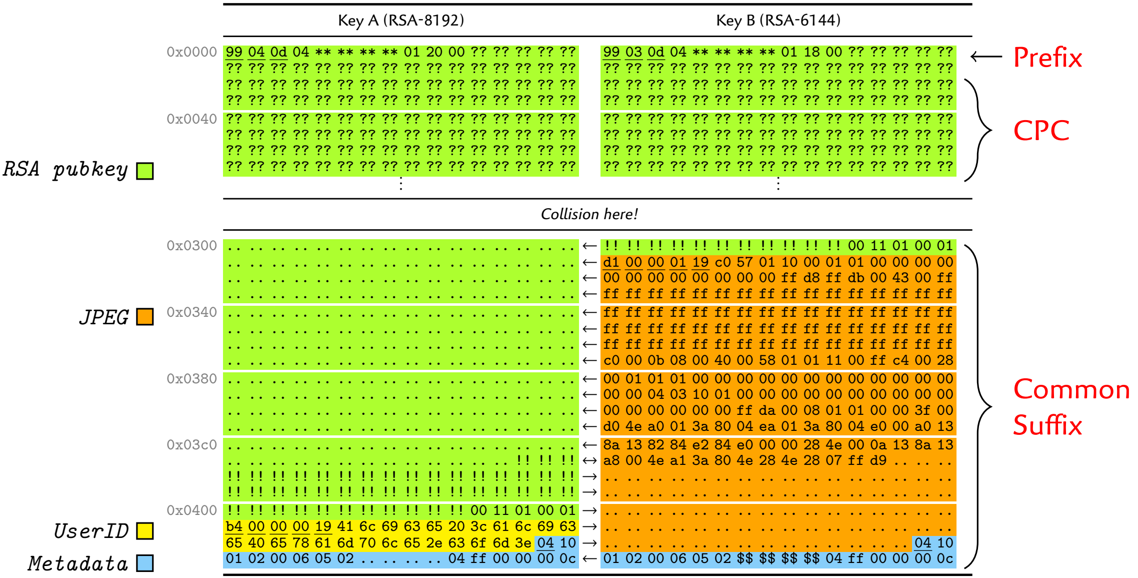
- PGP identity certificate has public key first, UserID next
  - Each blob prefixed by length
  - Cannot just use the ID a prefix as with X.509 certificates
  - Quite rigid format (weird extensions not signed)

- Use keys of different length, fields misaligned
- PGP format supports for JPEG picture in key, and picture can be signed
  - JPEG readers ignore garbage after End of Image marker

- Certificate A has RSA-8192 public key, with victim ID
- Certificate B has RSA-6144 public key, and attacker's picture
  - Stuff JPEG in key A, and ID B in JPEG
  - Need very small JPEG: example 181-byte JPEG (*almost compliant*)



Certificate structure



RSA pubkey 🟩

JPEG 🟧

UserID 🟨

Metadata 🟦

```
                           Key A (RSA-8192)                               Key B (RSA-6144)

0x0000  99 04 0d 04 ** ** ** ** 01 20 00 ?? ?? ?? ?? ??     99 03 0d 04 ** ** ** ** 01 18 00 ?? ?? ?? ?? ??   ← Prefix
        ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
        ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??   ┐
        ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??   │
0x0040  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??   │
        ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??   ├ CPC
        ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??   │
        ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??     ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??   ┘
                               ⋮                                                  ⋮

                                                    Collision here!

0x0300  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  !! !! !! !! !! !! !! !! !! !! !! 00 11 01 00 01   ┐
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  d1 00 00 01 19 c0 57 01 10 00 01 01 00 00 00 00   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  00 00 00 00 00 00 00 00 ff d8 ff db 00 43 00 ff   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   │
0x0340  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  c0 00 0b 08 00 40 00 58 01 01 11 00 ff c4 00 28   │
0x0380  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  00 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  00 00 04 03 10 01 00 00 00 00 00 00 00 00 00 00   │ Common
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  00 00 00 00 00 00 ff da 00 08 01 01 00 00 3f 00   ├ Suffix
        .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  d0 4e a0 01 3a 80 04 ea 01 3a 80 04 e0 00 a0 13   │
0x03c0  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..  ←  8a 13 82 84 e2 84 e0 00 00 28 4e 00 0a 13 8a 13   │
        .. .. .. .. .. .. .. .. .. .. .. .. .. !! !! !!  ↔  a8 00 4e a1 3a 80 4e 28 4e 28 07 ff d9 .. .. ..   │
        !! !! !! !! !! !! !! !! !! !! !! !! !! !! !! !!  →  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..   │
        !! !! !! !! !! !! !! !! !! !! !! !! !! !! !! !!  →  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..   │
0x0400  !! !! !! !! !! !! !! !! !! !! !! 00 11 01 00 01  →  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..   │
        b4 00 00 00 19 41 6c 69 63 65 20 3c 61 6c 69 63  →  .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..   │
        65 40 65 78 61 6d 70 6c 65 2e 63 6f 6d 3e 04 10  →  .. .. .. .. .. .. .. .. .. .. .. .. .. .. 04 10   │
        01 02 00 06 05 02 .. .. .. .. 04 ff 00 00 00 0c  ←  01 02 00 06 05 02 $$ $$ $$ $$ 04 ff 00 00 00 0c   ┘
```

Impersonation attack

1. Build CP collision with prefixes “`99040d04*012000`”/“`99030d04*011800`”
2. Choose JPEG image to include in B, UserID to include in A
3. Select “! !” bytes to make RSA modulus.
4. Ask for a signature of key B.
5. Copy the signature to key A.

- Single chosen-prefix collision can be used to target many victims
- Example keys on `https://sha-mbles.github.io`
  - Key creation date of our CPC in 2038 to avoid malicious usage

- GnuPGv1 (legacy branch) used `SHA-1` signatures by default
- Reported in May 2019, GnuPG stopped trusting `SHA-1` signatures (CVE-2019-14855)

SHA-1 Summary

`SHA-1` signatures can now be abused in practice

- `SHA-1` must be deprecated (same attacks as on `MD5` in 2007)
  - As long as `SHA-1` is supported, downgrade attacks are possible
  - Urgent for `SHA-1` signatures
    - SLOTH attack as long as `SHA-1` is supported in TLS, SSH [Bhargavan & L., NDSS'16]
    - Rogue CA using `SHA-1` X.509 certificates [Stevens *& al.*, C'09]

- GnuPGv2 stopped trusting `SHA-1` signatures (2019-11)
- Microsoft discontinued SHA-1 code signing support (2020-08)
- OpenSSH has disabled RSA-SHA1 signatures by default (2021-09)
- `SHA-1` deprecated for TLS in-protocol signatures (RFC9155 – 2021-12)

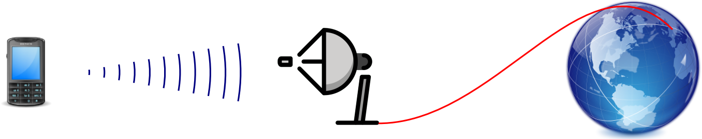
- Side result: breaking 64-bit crypto now costs less than 100 kUS$

Outline

GSM security

- A5/1 Cryptanalysis
- A5/2 Cryptanalysis

GSM Cell Phones



- GSM (2G) telephony first deployed in 1991
- GPRS is the data protocol of 2G telephony (sometimes called 2.5G)
  - Improved GPRS: EDGE (sometimes called 2.75G)
  - Designed by ETSI SAGE in 1998
- Widely used in the early 2000s
  - The first iPhone didn't support 3G (2008)
- 3G deployment: 2001–2010-ish
  - 2G has been sunset in some countries, but still used in France
  - Fallback when 3G/4G/5G not available
  - Used by some payment terminals

2G security

- Encryption of packets between the phone and the antenna
- Algorithms designed in secret in the 1980s and 1990s, not published

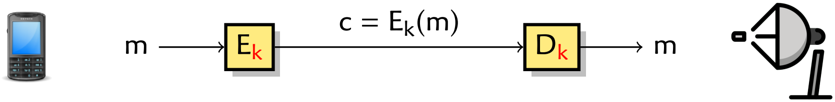
Voice: A5

A5/0	No encryption
A5/1	64-bit key, 64-bit state - Partial leak in 1994, Reverse engineered in 1999
A5/2	64-bit key, 81-bit state - Reverse engineered in 1999 - “export version” - Deprecated in 2007
A5/3	KASUMI with 64-bit key
A5/4	KASUMI with 128-bit key - Designed in 2002, public

Data: GEA (GPRS Encryption Algorithms)

GEA-0	No encryption
GEA-1	64-bit key, 96-bit state - Partial leak in 2011 [Nohl & Melette] - Deprecated in 2013
GEA-2	64-bit, 125-bit state
GEA-3	KASUMI with 64-bit key
GEA-4	KASUMI with 128-bit key - Designed in 2002, public

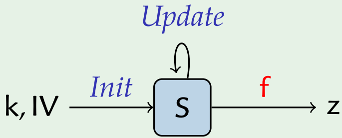
Stream ciphers



- Encrypt a message with a secret key k
- Keystream $z(k) = (z^{(0)}, z^{(1)}, z^{(2)}, \ldots)$
    - $c = E_k(m) = m \oplus z$

Stream cipher

- Internal state $S \in \mathcal{S}$
- State update function $\mathcal{S} \to \mathcal{S}$
- Extraction function $f : \mathcal{S} \to \{0,1\}$
- Initialization $k, IV \to \mathcal{S}$



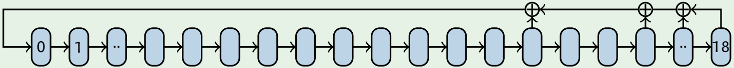
$$S^{(0)} = \mathrm{Init}(k) \qquad S^{(i+1)} = \mathrm{Update}(S^{(i)}) \qquad z^{(i)} = f(S^{(i)})$$

Linear Feedback Shift Register (LFSR)

- State S: n bits $(s_0, s_1, \dots, s_{n-1})$
- Linear update: $S^{(t+1)} = M \cdot S^{(t)}$

- Polynomial representation: $Q = X^n + \sum_{i \in \mathcal{A}} X^i$
  - If Q is primitive, update corresponds to multiplication by a primitive element
  - Maximal period if $S \neq 0$

Fibonacci configuration



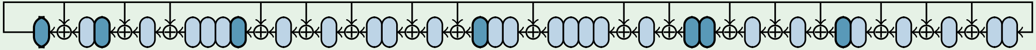
- Update depending on taps $\mathcal{A}$: $s_0^{(t+1)} = \sum_{i \in \mathcal{A}} s_i^{(t)}$, $s_{i+1}^{(t+1)} = s_i^{(t)}$

Linear Feedback Shift Register (LFSR)

- State S: n bits $(s_0, s_1, \dots, s_{n-1})$
- Linear update: $S^{(t+1)} = M \cdot S^{(t)}$

- Polynomial representation: $Q = X^n + \sum_{i \in \mathcal{A}} X^i$
  - If Q is primitive, update corresponds to multiplication by a primitive element
  - Maximal period if $S \neq 0$

Galois configuration

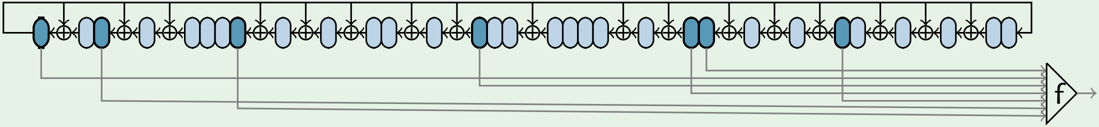


- Update depending on taps $\mathcal{A}$: $s_i^{(t+1)} = \begin{cases} s_{i+1}^{(t)} \oplus s_0^{(t)} & \text{if } i \in \mathcal{A} \\ s_{i+1}^{(t)} & \text{else} \end{cases}$

LFSR based stream ciphers

- Need to break linearity
  - Irregular clocking
  - Filter function of the state
  - Non-linear feedback

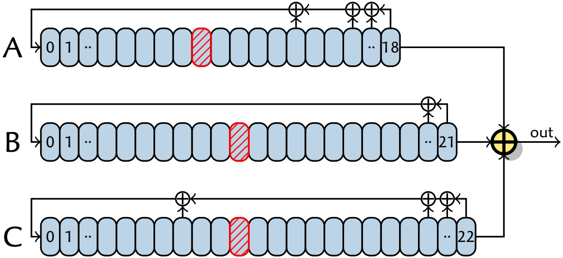
Filter generator



- Filter function to extract keystream from internal state (balanced, non-linear)
- Construction used in A5/1, A5/2, Bluetooth E0

A5/1

- Reverse engineered in 1999
- 3 LFSRs
  - A (19 bits)
  - B (22 bits)
  - C (23 bits)
- Irregular clocking:
  - $m = \mathrm{MAJ}(a_8, b_{10}, c_{10})$
  - Clock A iff $a_8 = m$
  - Clock B iff $b_{10} = m$
  - Clock C iff $c_{10} = m$



- The keystream is $z^{(i)} = a^{(i)}_{18} \oplus b^{(i)}_{21} \oplus c^{(i)}_{22}$
- Linear function of the state

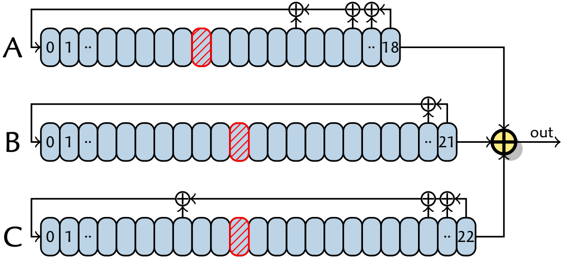
A5/1 initialization

Initialize the three LFSRs from 64-bit key and 22-bit frame number

1. Set A, B, C to zero
2. Clock them 64 + 22 times, xoring input bit into the feedback function
    - Clock registers always
3. Clock the register 100 times
    - Normal clocking dependant on registers content

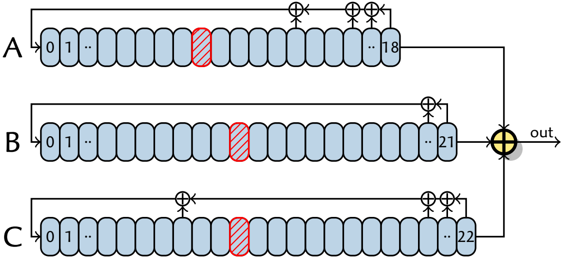
Security of A5/1

- Security: it should be hard to recover initial state from keystream



Security of A5/1

- **Security**: it should be hard to recover initial state from keystream



Main weakness

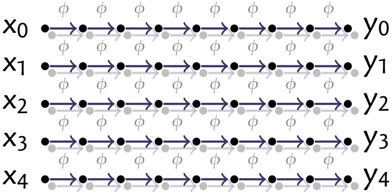
- State is too small (64 bits)

Time-memory tradeoff [Hellman, 1980]

- With known keystream z, invert public function $\phi : S \mapsto z^{(0)}, z^{(1)}, \dots, z^{(63)}$
- With precomputation: store $(\phi(S), S)$ indexed by $\phi(S)$
- Hellman tables: tradeoff with smaller storage size
  - Precomputation: N
  - Online: $TM^2 = N^2$

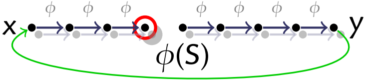
(Time T, Storage M, Domain size N)

1 **Precompute** iteration chain



2 **Store** (x_i, y_i)

3 **Online:** compute chain and restart



- In practice: precomputation too expensive
  - 2^{42} storage is 32 TB

Babbage-Golic time-memory tradeoff [Babbage, 1995] [Golic, 1997]

- With known keystream z, invert public function $\phi : S \mapsto z^{(0)}, z^{(1)}, \dots, z^{(63)}$
- Target one state out of many
  - $S^{(0)}$ produces keystream $z^{(0)}, z^{(1)}, z^{(2)}, \dots, z^{(n-1)}$
  - $S^{(1)}$ produces keystream $z^{(1)}, z^{(2)}, z^{(3)}, \dots, z^{(n)}$
  - $S^{(2)}$ produces keystream $z^{(2)}, z^{(3)}, z^{(4)}, \dots, z^{(n+1)}$

Meet-in-the-Middle attack / collision search

0 Capture frames with known plaintext, recover z

1 For 2^{32} random S, compute $\phi(S)$ and store in a hash table

2 For 2^{32} keystream prefixes z, look up z in the table

- In practice: 2^{32} keystreams takes too long to capture
  - Only 2^{22} keystreams in a two-minute call
  - → 2^{42} storage, or 2^{42} online time

Time-Memory-Data tradeoff [*Biryukov & Shamir, Asiacrypt'00*]

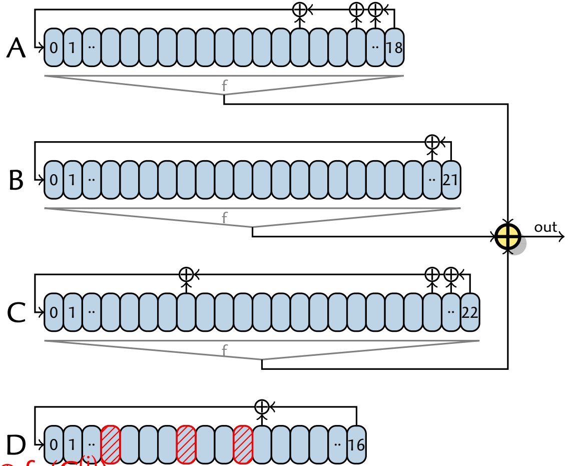
- Combine Hellman tables with Babbage-Golic time-memory tradeoff
  - Target one state out of many, precompute chains
- Better tradeoff than Hellman, because no need to cover full space

- Implemented in practice [Paget & Nohl, 2011]
  - Computed on GPU, ≈ 2TB storage
- There are known frames in GSM

Application to A5/1

- One frame gives 204 keystream prefixes
- Pre-computation $2^{64}/204 \approx 2^{57}$
- Storage 2^{37} ($\approx$ 1TB)
- Online cost: 2^{33}

A5/2



- Reverse engineered in 1999
- 4 LFSRs
  - A (19 bits)
  - B (22 bits)
  - C (23 bits)
  - D (17 bits)
- Clocking defined by D:
  - $m = \mathrm{MAJ}(d_{10}, d_3, d_7)$
  - Clock A iff $d_{10} = m$
  - Clock B iff $d_3 = m$
  - Clock C iff $d_7 = m$
- The keystream is $z^{(i)} = f_A(A^{(i)}) \oplus f_B(B^{(i)}) \oplus f_C(C^{(i)})$
- Non-linear function of the state, degree 2

  $z^{(i)} = a^{(i)}_{18} \oplus b^{(i)}_{21} \oplus c^{(i)}_{22} \oplus \mathrm{MAJ}(a^{(i)}_{15}, \bar{a}^{(i)}_{14}, a^{(i)}_{12}) \oplus \mathrm{MAJ}(\bar{b}^{(i)}_{20}, b^{(i)}_{13}, b^{(i)}_{9}) \oplus \mathrm{MAJ}(c^{(i)}_{22}, c^{(i)}_{20}, \bar{c}^{(i)}_{13})$

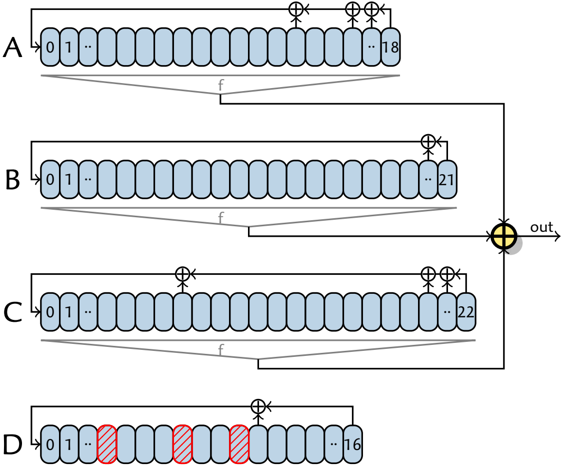
A5/2 initialization

Initialize the three LFSRs from 64-bit key and 22-bit frame number

1. Set A, B, C, D to zero
2. Clock them $64 + 22$ times, xoring input bit into the feedback function
    - Clock registers always
3. Set $a_{15} \leftarrow 1$, $b_{16} \leftarrow 1$, $c_{18} \leftarrow 1$, $d_{10} \leftarrow 1$
4. Clock the register 99 times
    - Normal clocking dependant on registers content

Security of A5/2

- Security: it should be hard to recover initial state from keystream

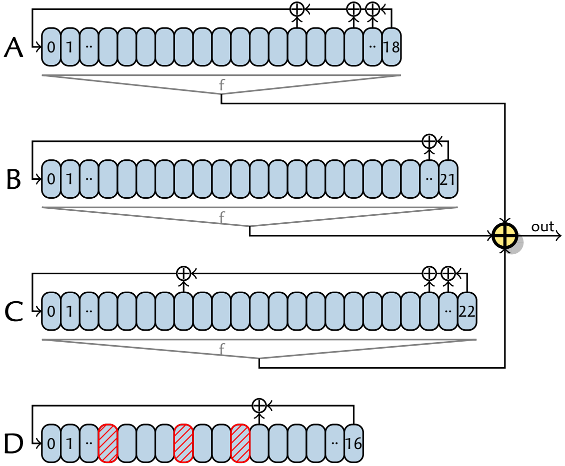


Security of A5/2

- Security: it should be hard to recover initial state from keystream

Main weakness

- Guessing D (16 bits) make clocking deterministic



Cryptanalysis of A5/2 [Goldberg, Wagner & Green, '99]

1. Consider two frames with distance 2^{11}
   - Difference in D absorbed by $d_{10} \leftarrow 1$
   - Known difference in A, B, C
2. Guess initial state of D
   - All clocking become known
   - State differences known at all clocks by linearity
3. Keystream difference is a linear function of initial state
   - $A \mapsto f(A) \oplus f(A \oplus \delta)$ is a derivative of f
   - Since f has the degree two, the derivative is linear

- Complexity: 2^{16} dot-products (linear functions)

Semi-active downgrade attack [Barkan, Biham & Keller, C'03]

- Passive: Record frames encrypted with strong cipher (A5/1, A5/3, ...)
- Active: force phone to use A5/2 with same key, recover key

A5/1 and A5/2 Summary

- A5/1 broken in practice because state is too small (64 bits)
  - Practical (low data) with large precomputation (2^{56})
- A5/2 much weaker
  - Using a separate register for clocking weakens the cipher

Export ciphers

- A5/2 was designed to use GSM in countries with export regulations of crypto
- First implementations of GSM used only 56-bit session keys
- Other examples of "export" ciphersuites in TLS

- A5/2 design document states: [ETR 278]
  "The algorithm must be such that export controls in force in a number of CEPT member countries permit its use in accordance with the GSM MoU policy reproduced in annex A"

Outline

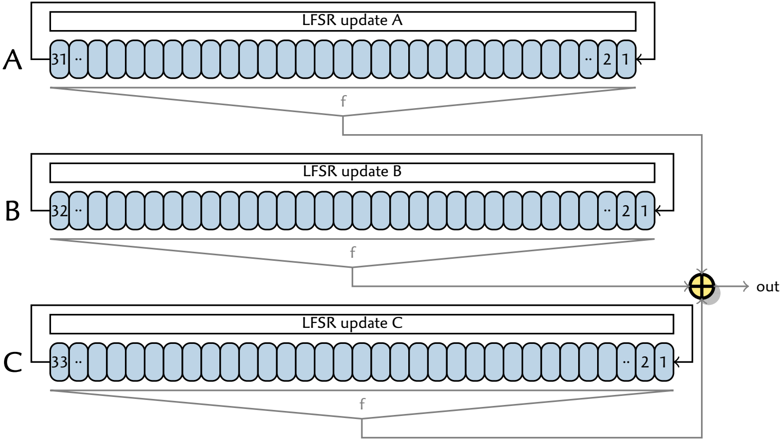
GPRS Encryption

- GEA-1 Cryptanalysis
- GEA-2 Cryptanalysis

C. Beierle, P. Derbez, G. Leander, G. L., H. Raddum, Y. Rotella, D. Rupprecht, L. Stennes
Cryptanalysis of the GPRS Encryption Algorithms GEA-1 and GEA-2
Eurocrypt 2020

GEA-1 design

- Received specification from anonymous source
- Three filter generators
  - A (31 bits) ↪ $\texttt{Gen}_A(A)$
  - B (32 bits) ↪ $\texttt{Gen}_B(B)$
  - C (33 bits) ↪ $\texttt{Gen}_C(C)$
- Non-linear filtering
  - degree-4 function f



- The keystream is $z = \texttt{Gen}_A(A) \oplus \texttt{Gen}_B(B) \oplus \texttt{Gen}_C(C)$

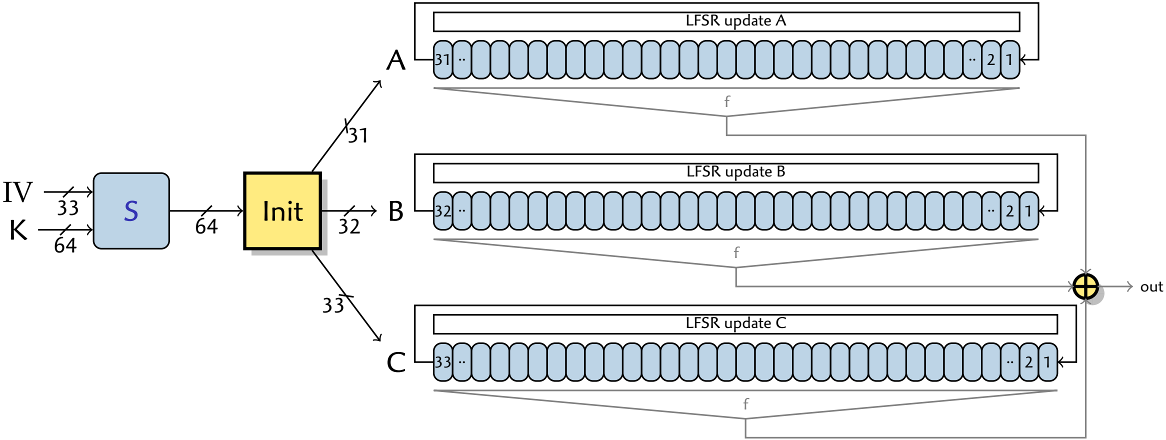
GEA-1 initialization

1. Generate a 64-bit value S from the key and IV
    - Using a NLFSR (non linear)
2. Initialize the three LFSRs from S
    - Set A, B, C to zero
    - Clock them 64 times, xor s_i into the feedback function
        - A uses $s_0\ , s_1\ , \dots, s_{64}$
        - B uses $s_{16}, s_{17}, \dots, s_{15}$ (shifted by 16 positions)
        - C uses $s_{32}, s_{33}, \dots, s_{31}$ (shifted by 32 positions)

    - If register is zero, set to one (ignored in our analysis).

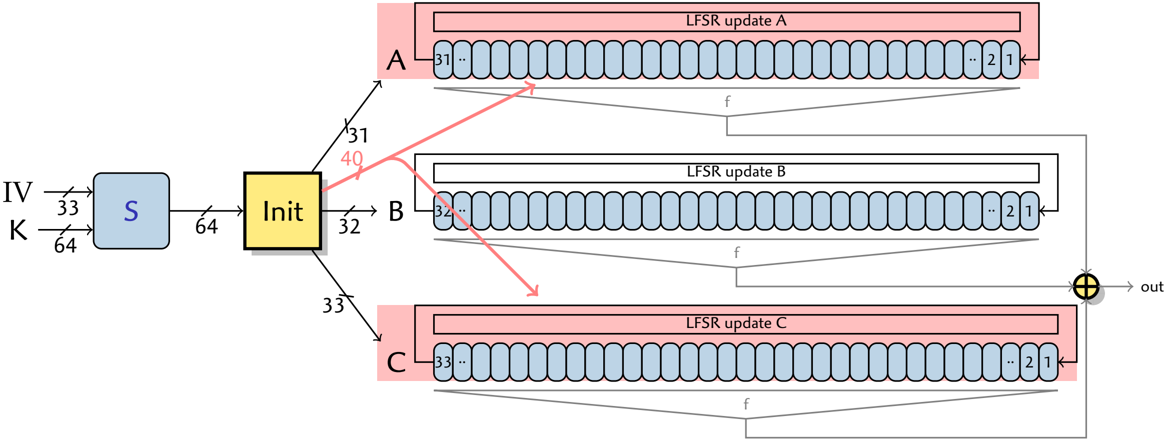
- Initialization of A, B, C from S is linear
    - $S \mapsto A$: 64 bit → 31 bits, rank 31
    - $S \mapsto B$: 64 bit → 32 bits, rank 32
    - $S \mapsto C$: 64 bit → 33 bits, rank 33
    - $S \mapsto (A, B, C)$: 64 bit → 96 bits, rank 64
    - $S \mapsto (A, C)$: 64 bit → 64 bits, rank 40

GEA-1 initialization



- Initialization of A, B, C from S is linear
  - $S \mapsto A$: 64 bit → 31 bits, rank 31
  - $S \mapsto B$: 64 bit → 32 bits, rank 32
  - $S \mapsto C$: 64 bit → 33 bits, rank 33
  - $S \mapsto (A, B, C)$: 64 bit → 96 bits, rank 64
  - $S \mapsto (A, C)$: 64 bit → 64 bits, rank 40

GEA-1 initialization



- Initialization of A, B, C from S is linear
  - $S \mapsto A$: 64 bit → 31 bits, rank 31
  - $S \mapsto B$: 64 bit → 32 bits, rank 32
  - $S \mapsto C$: 64 bit → 33 bits, rank 33
  - $S \mapsto (A, B, C)$: 64 bit → 96 bits, rank 64
  - $S \mapsto (A, C)$: 64 bit → 64 bits, rank 40

Meet-in-the-Middle attack

- There are 2^{40} possible initial states for (A, C)
- There are 2^{32} possible initial states for B
- The keystream is $z = \mathtt{Gen}_A(A) \oplus \mathtt{Gen}_B(B) \oplus \mathtt{Gen}_C(C)$
  - Split in two independent parts: $\mathtt{Gen}_B(B) = z \oplus \mathtt{Gen}_A(A) \oplus \mathtt{Gen}_C(C)$

Meet-in-the-Middle attack / collision search

0. Capture frame with known plaintext, recover z
1. For all 2^{32} B, compute $\mathtt{Gen}_B(B)$ and store in a hash table
2. For all 2^{40} (A, C), compute $z \oplus \mathtt{Gen}_A(A) \oplus \mathtt{Gen}_C(C)$ and look up in the table

- Recover the key from the initial state (A, B, C)
- Complexity
  - 64 bits of known keystream
  - 2^{40} Time
  - 2^{32} Memory

Reducing memory

- Memory usage can be reduced significantly [Amzaleg & Dinur, EC'22]
- Reduce memory usage from 2^{32} to 2^{24}
  - (A, C) and (B) are not independent
  - Start by guessing 8 common bits of information
- Further reduce to 2^{19} (4MB) using techniques from 3-XOR cryptanalysis

Backdoor?

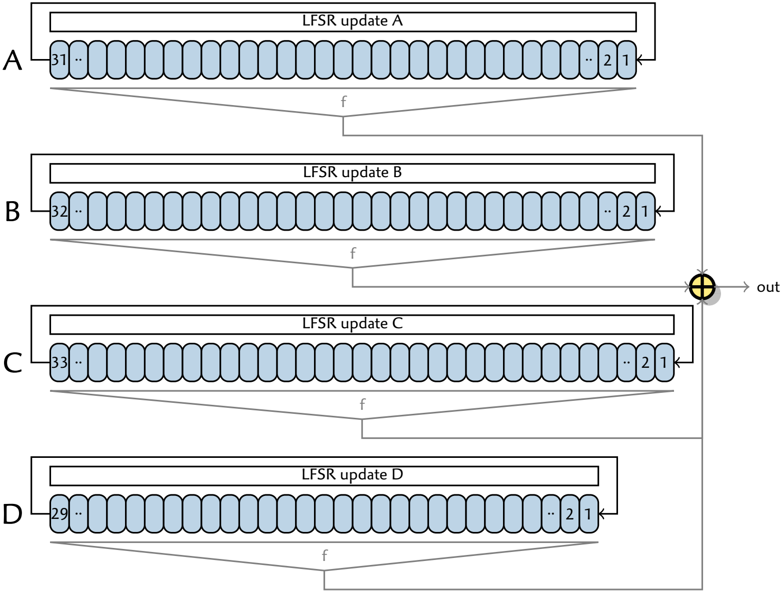
GEA-1 was likely weakened deliberately

- Mapping $S \mapsto A, C$ from 64 bits to 64 bits
  - Having rank 40 is very unlikely
- Experiments with initialization of the same type
  - With 1 million experiments, lowest rank found is 55
  - Follow-up work to build LFSRs and shift with low rank [Beierle, Felke & Leander, 2021]

- In the 1990's, cryptography was subjected to export regulation
  - In France, 40-bit security cryptography can be exported after 1998
- The design document states:
  "the algorithm should be generally exportable taking into account current export restrictions"
  "the strength should be optimized taking into account the above requirement"
- Other examples of "export" ciphersuites: TLS, A5/2 in GSM

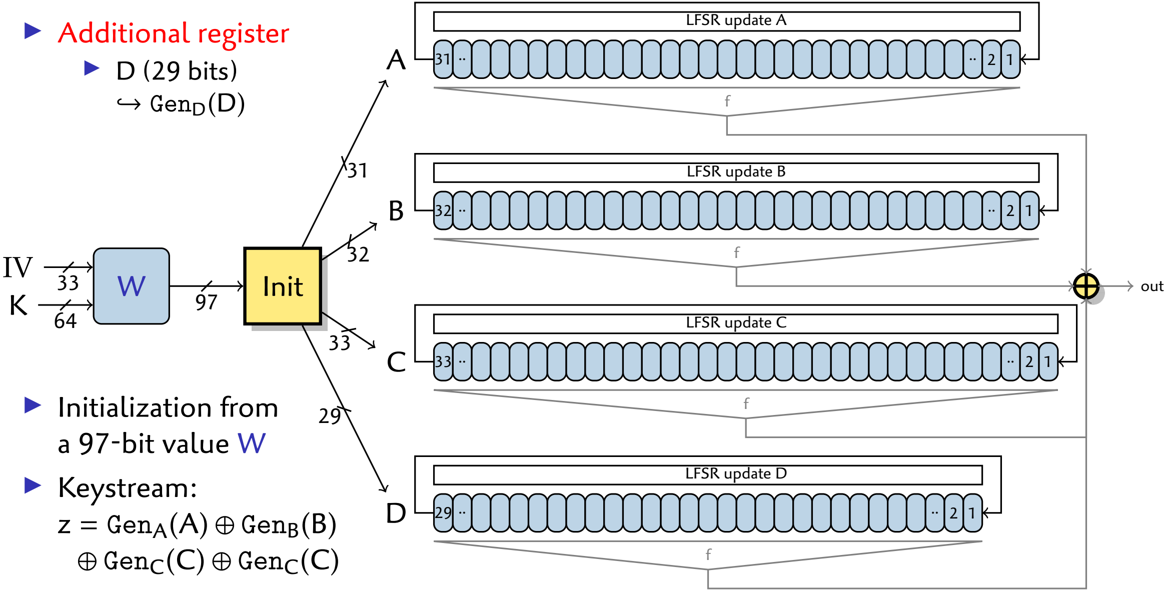
GEA-2 design

- Additional register
  - D (29 bits)
    $\hookrightarrow \mathtt{Gen}_D(D)$
- Initialization from a 97-bit value W
- Keystream:
  $z = \mathtt{Gen}_A(A) \oplus \mathtt{Gen}_B(B)$
  $\oplus \mathtt{Gen}_C(C) \oplus \mathtt{Gen}_C(C)$



GEA-2 design

- Additional register
  - D (29 bits)
    $\hookrightarrow \mathtt{Gen}_D(D)$



- Initialization from a 97-bit value W
- Keystream:
  $z = \mathtt{Gen}_A(A) \oplus \mathtt{Gen}_B(B)$
  $\oplus\, \mathtt{Gen}_C(C) \oplus \mathtt{Gen}_C(C)$

Meet-in-the-Middle attack

- The keystream is $z = \texttt{Gen}_A(A) \oplus \texttt{Gen}_B(B) \oplus \texttt{Gen}_C(C) \oplus \texttt{Gen}_D(D)$
  - Register sizes: 31 (A), 32 (B), 33(C), 29 (D)
- Standard MitM: $\texttt{Gen}_A(A) \oplus \texttt{Gen}_B(B) = z \oplus \texttt{Gen}_C(C) \oplus \texttt{Gen}_D(D)$
  - Complexity $\approx 2^{63}$ ((A, B) is 63 bits, (C, D) is 62 bits)
- No unexpected rank loss

Algebraic attack: linearisation

Writing $z^{(i)} = \mathtt{Gen}_A^{(i)}(A) \oplus \mathtt{Gen}_B^{(i)}(B) \oplus \mathtt{Gen}_C^{(i)}(C) \oplus \mathtt{Gen}_D^{(i)}(D)$ *as a polynomial*

- $31 + 32 + 33 + 29 =$ 125 variables
- Each keystream bit $z^{(i)}$ gives an equation

Toy example

- Small number of possible monomials
  - LFSR update is linear
  - The filtering function f has algebraic degree 4
  - $\sum_{i=1}^{4} \binom{31}{i} + \binom{32}{i} + \binom{33}{i} + \binom{29}{i} =$ 152682 monomials

- Linearisation attack:
  - Consider each monomial as an independent variable
  - Solve the linear system
  - Complexity $152682^3 \approx 2^{52}$
- Requires about 152682 bits of keystream z
- Problem: GPRS frame is at most 12800 bits

Partial guessing

- We can reduce the number of monomial below 12800 by guessing some state bits
- For instance: guess 15 bits of A, 15 bits of B, 16 bits of C, 13 bits of D
  - Remaining variables: 16 (A) + 17 (B) + 17 (C) + 16 (D)
  - $\sum_{i=1}^{4} \binom{16}{i} + \binom{17}{i} + \binom{17}{i} + \binom{16}{i} = 11468$ monomials (< 12800)
- Solve the remaining system with linear algebra
  - Complexity $\approx 2^{59} \times 12800^3$

Hybrid Meet-in-the-Middle

Strategy

1. Guess parts of A and D
2. Find relations that depend only on B, C: $\phi(B) \oplus \psi(C) = \xi(z)$

- Guess 11 bits of A and 9 bits of D
- Write $w^{(i)} = \mathtt{Gen}_A^{(i)}(A) \oplus \mathtt{Gen}_D^{(i)}(D)$ as a polynomial in the remaining variables (20+20)
- Look for masks m (length 12800) such that $m \cdot w_0 \dots w_{12799}$ is constant
  - $\sum_{i=1}^{4} \binom{20}{i} + \binom{20}{i} = 12390$ non-constant monomials
  - Using linearisation, space of good masks of dimension $12800 - 12390 = 410$
- Build linear function L from 64 independent masks:
  - $z = \mathtt{Gen}_D(D) \oplus \mathtt{Gen}_A(A) \oplus \mathtt{Gen}_B(B) \oplus \mathtt{Gen}_C(C)$
  - $\underbrace{L(z)}_{\text{known}} = \underbrace{L(\mathtt{Gen}_D(D)) \oplus L(\mathtt{Gen}_A(A))}_{\text{constant}} \oplus \underbrace{L(\mathtt{Gen}_B(B))}_{\phi(B)} \oplus \underbrace{L(\mathtt{Gen}_C(C))}_{\psi(C)}$

Linearization: toy example

$$
\begin{array}{rcllllllllll}
 & & 1 & a_0 & a_1 & a_2 & a_0a_1 & a_0a_2 & a_1a_2 & b_0 & b_1 & b_0b_1 \\
w_0 & = & 1\oplus & a_0\oplus & & & & & & b_0 & & \\
w_1 & = & & & a_1\oplus & & & a_0a_2\oplus & & & b_1\oplus & b_0b_1 \\
w_2 & = & 1\oplus & a_0\oplus & & a_2\oplus & a_0a_1\oplus & & & & & b_0b_1 \\
w_3 & = & 1\oplus & a_0\oplus & a_1\oplus & & a_0a_1\oplus & & a_1a_2\oplus & b_0\oplus & b_1 & \\
w_4 & = & & & & a_2\oplus & & a_0a_2\oplus & & b_0\oplus & & b_0b_1 \\
w_5 & = & & a_0\oplus & & a_2\oplus & & & a_1a_2\oplus & & b_1\oplus & b_0b_1 \\
w_6 & = & & & a_1\oplus & & a_0a_1\oplus & a_0a_2\oplus & & b_0 & & \\
w_7 & = & 1\oplus & a_0\oplus & a_1\oplus & & a_0a_1\oplus & & a_1a_2\oplus & & & b_0b_1 \\
w_8 & = & 1\oplus & a_0\oplus & & a_2\oplus & & & a_1a_2\oplus & & b_1\oplus & b_0b_1 \\
w_9 & = & & & a_1\oplus & a_2\oplus & & a_0a_2\oplus & & b_0\oplus & b_1\oplus & b_0b_1 \\
w_{10} & = & & & a_1\oplus & & a_0a_1\oplus & a_0a_2\oplus & & & b_1 & \\
w_{11} & = & & a_0\oplus & a_1\oplus & & & & & & b_1\oplus & b_0b_1 \\
\hline
w_0 \oplus w_2 \oplus w_9 \oplus w_{10} & = & 1 & & & & & & & & & \\
w_2 \oplus w_5 \oplus w_7 \oplus w_{11} & = & 0 & & & & & & & & & \\
w_5 \oplus w_8 & = & 1 & & & & & & & & &
\end{array}
$$

Hybrid Meet-in-the-Middle

Precomputation

- For each 2^{20} (a, d) (partial guess of A and D)
  1. Compute linear combinations of w independent of remaining (A, D)
  2. Deduce functions $\phi_{a,d}, \psi_{a,d}, \xi_{a,d}$ such that $\phi_{a,d}(B) = \psi_{a,d}(C) \oplus \xi_{a,d}(z)$

- Complexity: $2^{20} \times 12800^3/64 \approx 2^{54.9}$ 64-bit operations

Meet-in-the-Middle attack / collision search

- For each 2^{20} (a, d) (partial guess of A and D)
  1. For all 2^{32} B, compute $\phi_{a,d}(B)$ and store in a hash table
  2. For all 2^{33} C, compute $\xi_{a,d}(z) \oplus \psi_{a,d}(C)$ and look up in the table
     - If there is match, recover key candidate from a, B, C, d

- Evaluation of $\phi_{a,d}, \psi_{a,d}$ as polynomials with amortized cost 4 [BCCCNSY, CHES'10]
- Complexity: $2^{52} + 2^{53} \approx 2^{53.6}$ memory access; $2^{54} + 2^{55} \approx 2^{55.6}$ 64-bit operations

Improvement: Time-Data Tradeoff

- Classical technique: target one state out of many [Babbage, 1995] [Golic, 1997]
- We target the first 753 states; 753 keystreams of length 12047
  - $(A^{(0)}, B^{(0)}, C^{(0)}, D^{(0)})$ produces keystream $z^{(0)}z^{(1)}z^{(2)} \dots$
  - $(A^{(1)}, B^{(1)}, C^{(1)}, D^{(1)})$ produces keystream $z^{(1)}z^{(2)}z^{(3)} \dots$
  - $(A^{(2)}, B^{(2)}, C^{(2)}, D^{(2)})$ produces keystream $z^{(2)}z^{(3)}z^{(4)} \dots$
- Guess 11 bits of A and 10 bits of D
  - Write $w^{(i)} = \mathtt{Gen}_A^{(i)}(A) \oplus \mathtt{Gen}_D^{(i)}(D)$ as a polynomial in the remaining variables (19+20)
- Look for masks m (length 12047) such that $m \cdot w^{(0)} \dots w^{(12046)}$ is constant
  - $\sum_{i=1}^{4} \binom{19}{i} + \binom{20}{i} = 11230$ non-constant monomials
  - Using linearisation, space of good masks of dimension $12047 - 11230 = 817$
- Filter masks such that $m \cdot z^{(0)} \dots z^{(12046)} = m \cdot z^{(1)} \dots z^{(12047)} = m \cdot z^{(2)} \dots z^{(12048)} = \cdots$
  - Space of good masks of dimension $817 - 752 = 65$ (752 constraints)
- Build linear function L from 64 independent masks:
  - $z^{(s)}z^{(s+1)} \dots = \mathtt{Gen}_D(D^{(s)}) \oplus \mathtt{Gen}_A(A^{(s)}) \oplus \mathtt{Gen}_B(B^{(s)}) \oplus \mathtt{Gen}_C(C^{(s)})$
  - $\underbrace{L(z^{(s)}z^{(s+1)} \dots)}_{\text{independent of } s} = \underbrace{L(\mathtt{Gen}_D(D^{(s)})) \oplus L(\mathtt{Gen}_A(A^{(s)}))}_{\text{constant}} \oplus \underbrace{L(\mathtt{Gen}_B(B^{(s)}))}_{\phi(B^{(s)})} \oplus \underbrace{L(\mathtt{Gen}_C(C^{(s)}))}_{\psi(C^{(s)})}$

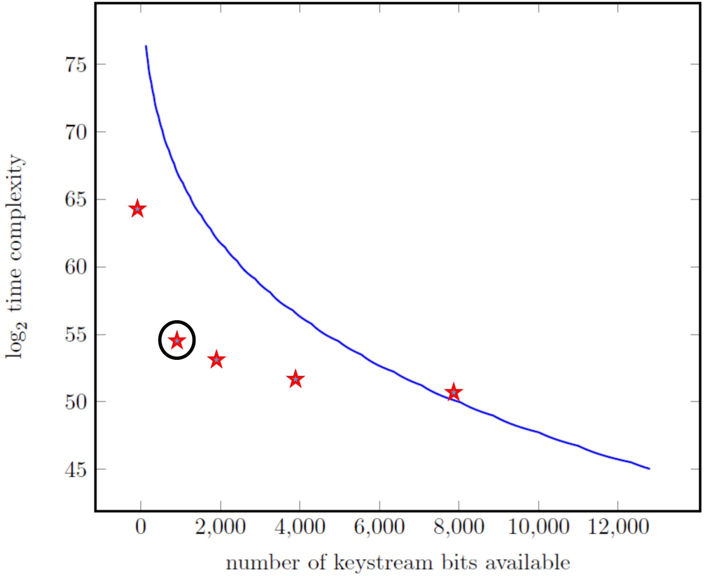
Hybrid Meet-in-the-Middle with Time-Data Tradeoff

Meet-in-the-Middle attack / collision search

- For each 2^{21} (a, d) (partial guess of A and D)
  0. Build functions $\phi_{a,d}, \psi_{a,d}, \xi_{a,d}$ such that $\phi_{a,d}(B) \oplus \psi_{a,d}(C) = \xi_{a,d}(z_s z_{s+1} \ldots)$
  1. For all 2^{32} B, compute $\phi_{a,d}(B)$ and store in a hash table
  2. For all 2^{33} C, compute $\xi_{a,d}(z) \oplus \psi_{a,d}(C)$ and look up in table
     - If there is match, recover key candidate from a, B, C, d

- On average, only $2^{21}/753 \approx 2^{11.4}$ guesses until it matches one of the 753 targets
- Complexity: $2^{11.4} \times 2^{33.6} \approx 2^{45}$ memory access; $4 \times 2^{45} \approx 2^{47}$ 64-bit operations

Time-data tradeoff



- Complexity 2^{45} with full frame (12800 bits)
- Tradeoff with fewer data *(blue line)*
- Better tradeoff with different attack: 4XOR *(stars)* [Amzaleg & Dinur, EC'22]

Usage and deprecation

- In 2011, large usage of GEA-1 and GEA-2 [Nohl & Melette]
- GEA-1 deprecated in 2013

- In 2021, large usage of GEA-3 (also GEA-0 🤯) [umlaut report]
  - Some operators use GEA-2 as main algorithm
  - One operator seen using GEA-1 sometimes

- GEA-1 still implemented in recent phones!
  - (iPhone 8, Galaxy S9, ...)

- We contacted GSMA and ETSI for responsible disclosure
  - New test-case to verify non-implementation of GEA-1
  - Plans to deprecate GEA-2

GEA-1 and GEA-2 Summary

- GEA-1 attack completely practical
  - Only 64 bits of known keystream, 2^{40} operations
  - 2.5 hours on a laptop today, practical in the 2000's

- GEA-2 attack borderline practical
  - Full frame known (12800 bits), 2^{45} operations
  - 4 months on a server

- In the early 2000's, internet traffic was mostly in the clear (low TLS use)
- Today, breaking GEA gives some metadata

- Semi-active downgrade attack [Barkan, Biham & Keller, C'03]
  - Passive: Record frames encrypted with GEA-3
  - Active: force phone to use GEA-1 with same key, recover key

GEA-1 and GEA-2 Summary

- GEA-1 attack completely practical
    - Only 64 bits of known keystream, 2^{40} operations
    - 2.5 hours on a laptop today, practical in the 2000's
- GEA-2 attack borderline practical
    - Full frame known (12800 bits), 2^{45} operations
    - 4 months on a server
- In the early 2000's, internet traffic was mostly in the clear (low TLS use)
- Today, breaking GEA gives some metadata
- Semi-active downgrade attack [Barkan, Biham & Keller, C'03]
    - Passive: Record frames encrypted with GEA-3
    - Active: force phone to use GEA-1 with same key, recover key

Conclusion

- Cryptography is usually a strong basis for security,
  but we need public cryptanalysis to assess primitives

- Security by obscurity does not work
  - A5/1
  - A5/2
  - GEA-1
  - GEA-2
  - Mifare
  - Keeloq
  - DVDCSS
  - ...

- Broken ciphers must be deprecated as soon as possible
  - RC4
  - MD5
  - `SHA-1`
- Demonstration of practical attacks helps

- Mismatch between security assumption and primitive choice
  - Security models, data limits, ...

- Backdoors affect the security of everybody
  - GEA-1 used outside “export” countries
  - Downgrade attack as long as weak algorithm are implemented
  - Other example: Logjam, exploiting TLS “export” ciphersuites