Extracting the RC4 secret key of the Open Smart Grid Protocol (OSGP)

Industrial Control System
Security Workshop (ICSS)

Los Angeles, USA
8 December 2015

Linus Feiten and Matthias Sauer

University of Freiburg
Germany

Chair of
Computer Architecture

UNI
FREIBURG

Outline

Preliminaries

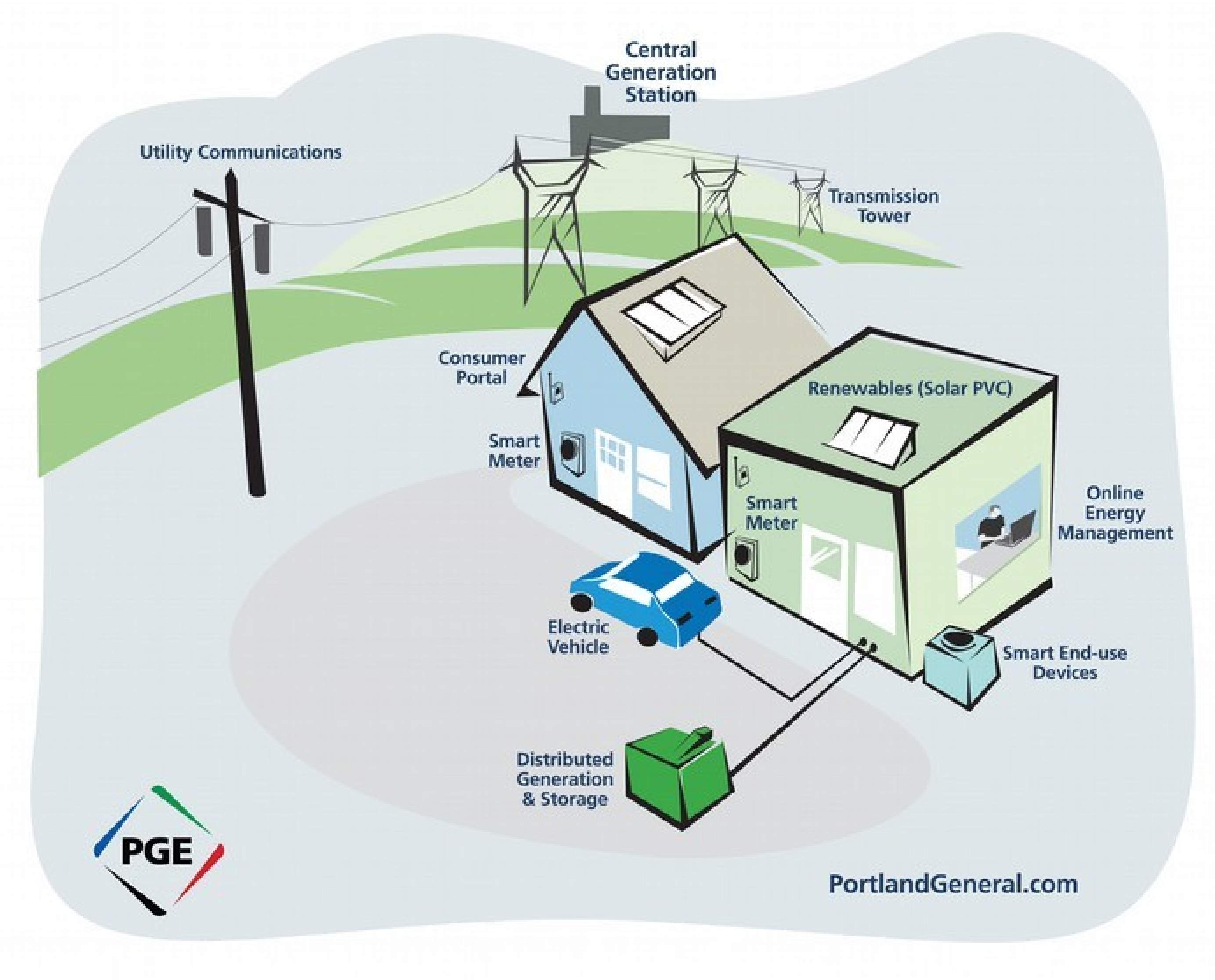
- Smart Grid
- Open Smart Grid Protocol (OSGP)
- Security in OSGP

Attack on OSGP data confidentialty

- Weakness of classical RC4
- Adapting attack to OSGP's RC4
- Practicality of attack

Countermeasures?

The Smart Grid



(Copyright: Portland General Electric, "Smart Grid" via Flickr, Creative Commons Attribution-NoDerivs 2.0 Generic)

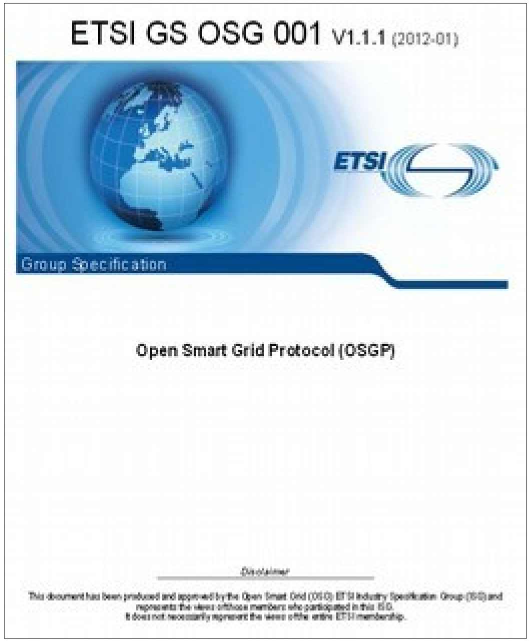
Open Smart Grid Protocol (OSGP)

Development started 2010.

Maintained by OSGP Alliance (www.osgp.org) and
Networked Energy Services Corp (www.networkedenergy.com).

Since 2012 freely available as European Telecommunications Standards Institute (ETSI) specification GS OSG 001.

Over 3.5 million devices worldwide.



ETSI GS OSG 001 V1.1.1 (2012-01)

Group Specification

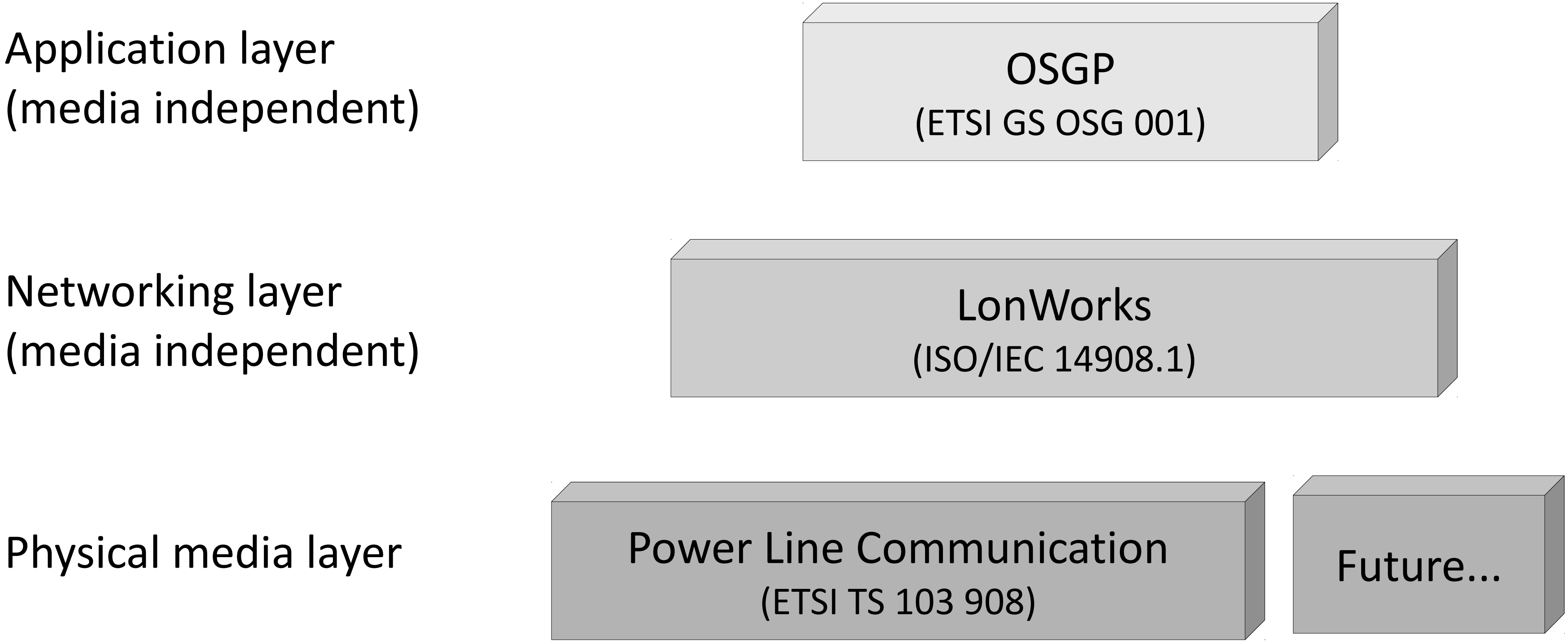
Open Smart Grid Protocol (OSGP)

Disclaimer

This document has been produced and approved by the Open Smart Grid (OSG) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

(http://www.etsi.org/deliver/etsi_gs/osg/001_099/001/01.01.01_60/gs_osg001v010101p.pdf)

OSGP communication network



Application layer
(media independent)

OSGP
(ETSI GS OSG 001)

Networking layer
(media independent)

LonWorks
(ISO/IEC 14908.1)

Physical media layer

Power Line Communication
(ETSI TS 103 908)

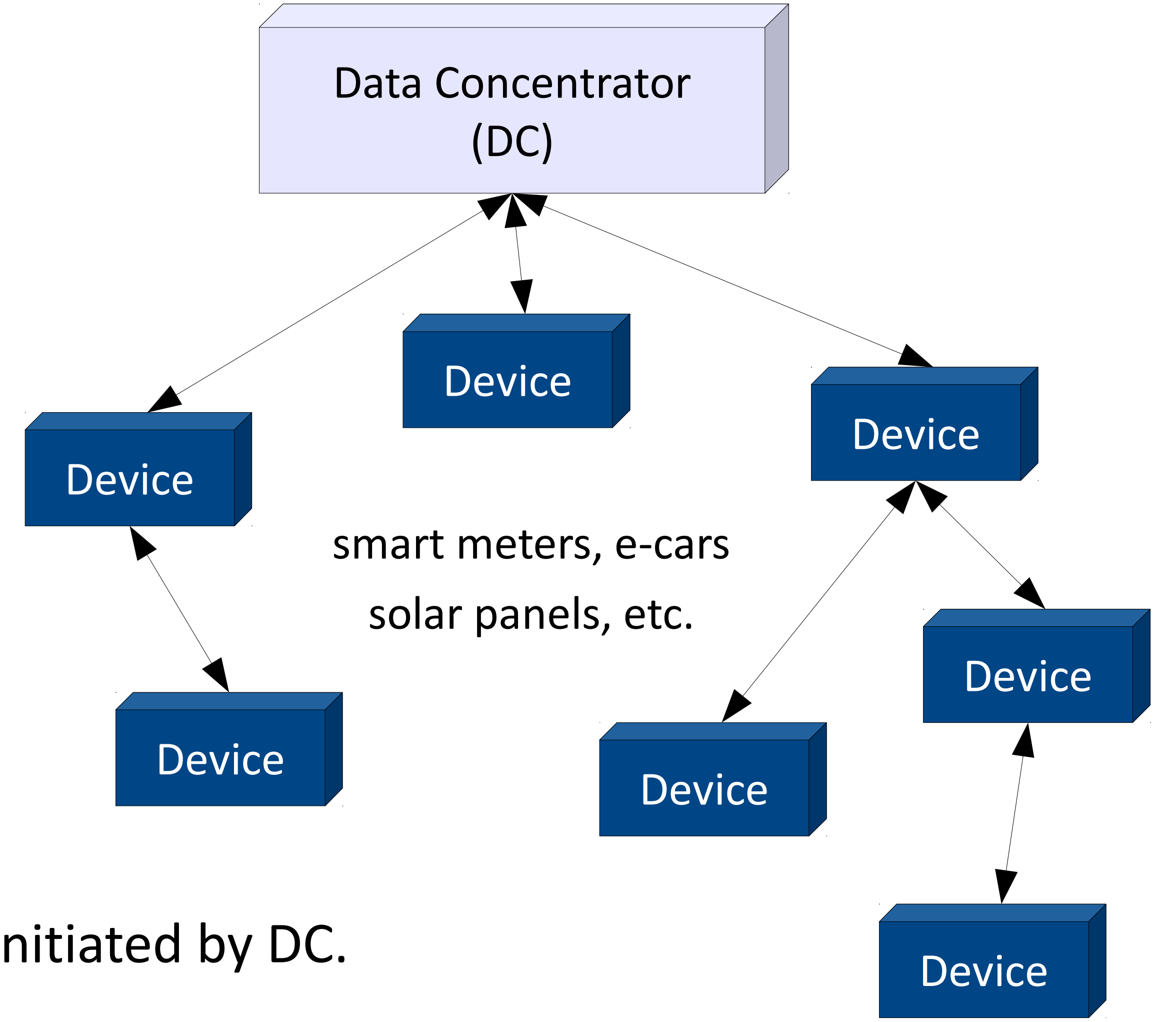
Future...

OSGP communication network

Power plant or grid operator
(load balancing, billing).

Electricity “prosumers”.

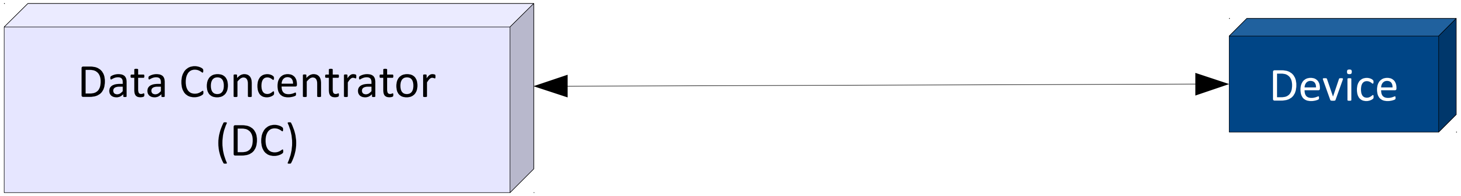
OSGP devices also act
as message repeaters.



smart meters, e-cars
solar panels, etc.

Master/slave communication only initiated by DC.

OSGP communication security



Data integrity

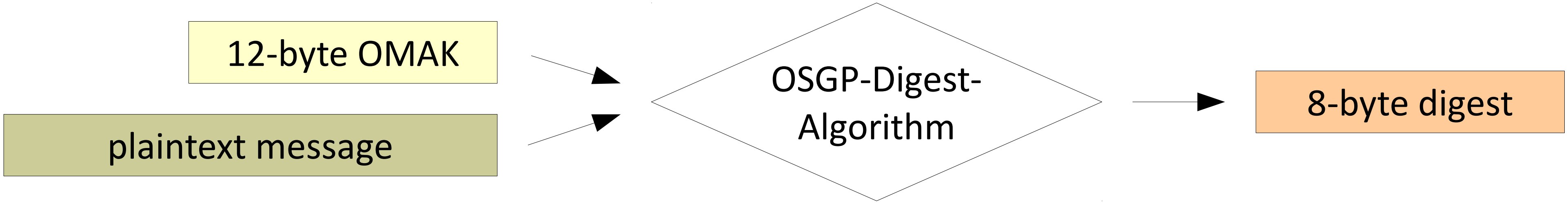
Prevent man-in-the-middle from forging data, e.g. switch on/off devices.

Data confidentiality

Prevent eavesdropper from reading sensitive data on electricity consumption.

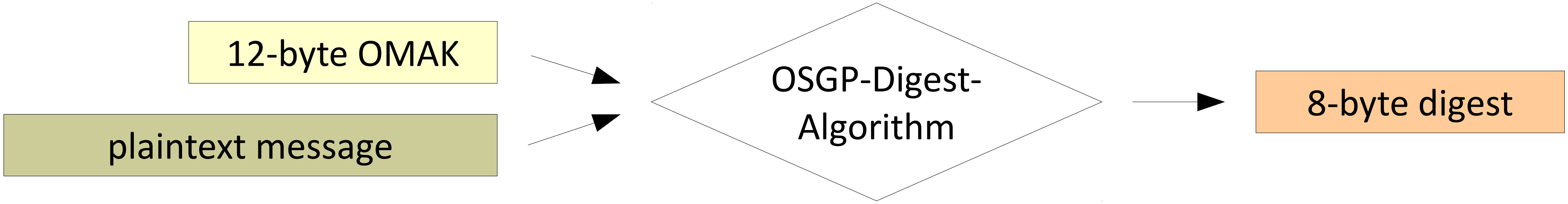
OSGP data integrity

For each message, generate a ***digest*** (hash value) using the secret "Open Media Access Key" (***OMAK***):

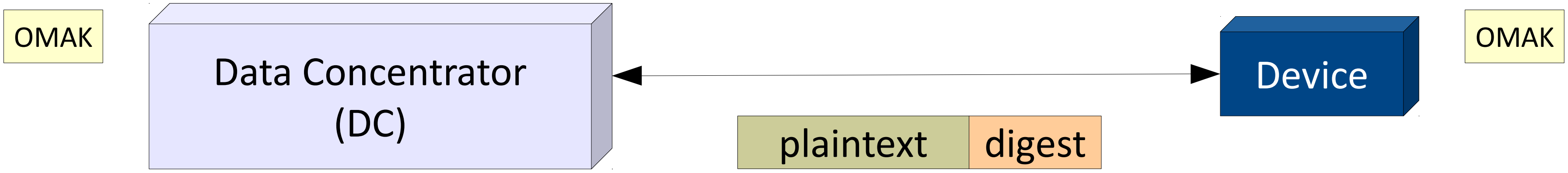


OSGP data integrity

For each message, generate a ***digest*** (hash value) using the secret “Open Media Access Key” (***OMAK***):

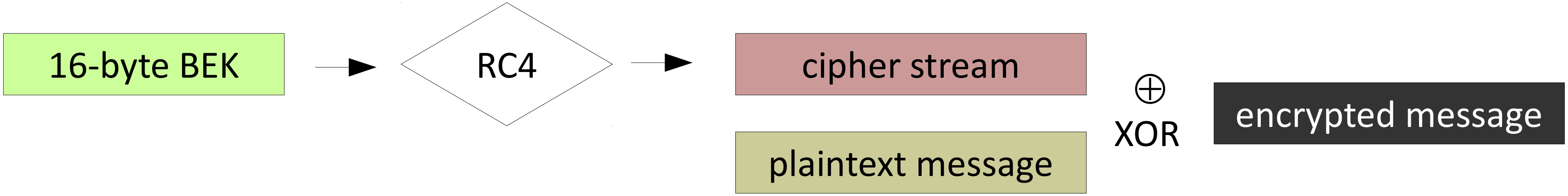


Transmit message and its digest:



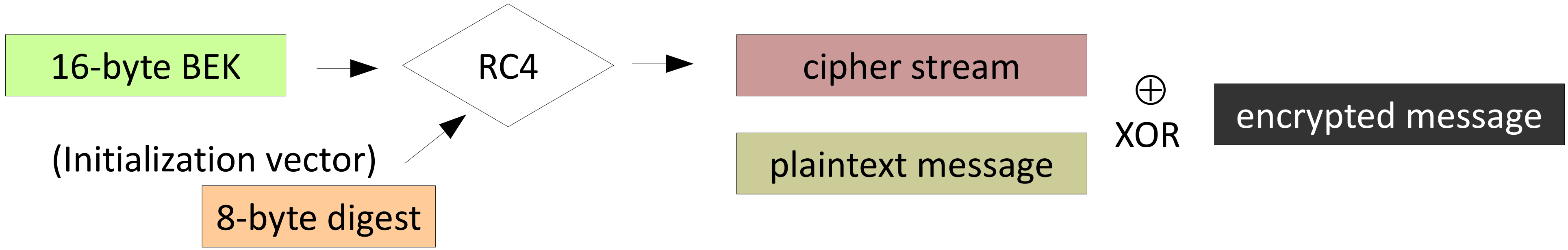
OSGP data confidentiality

For each message, generate an RC4 cipher stream using the secret “Base Encryption Key” (***BEK***). Encryption by xor of plaintext and cipher stream.



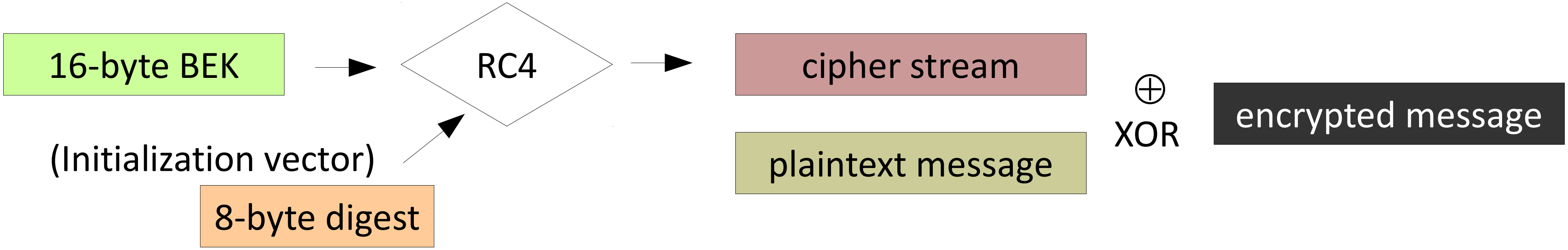
OSGP data confidentiality

For each message, generate an RC4 cipher stream using the secret “Base Encryption Key” (***BEK***). Encryption by xor of plaintext and cipher stream.

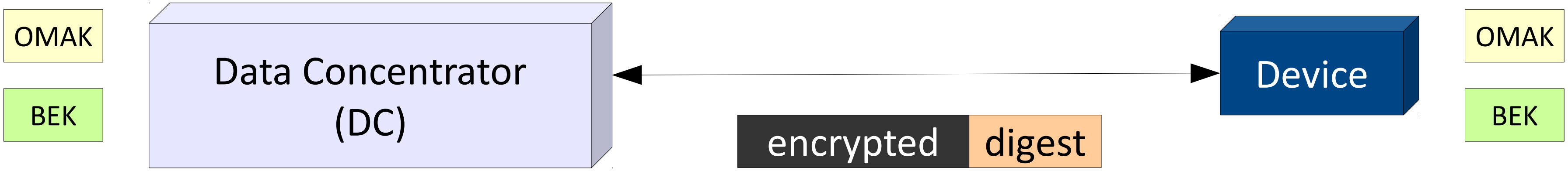


OSGP data confidentiality

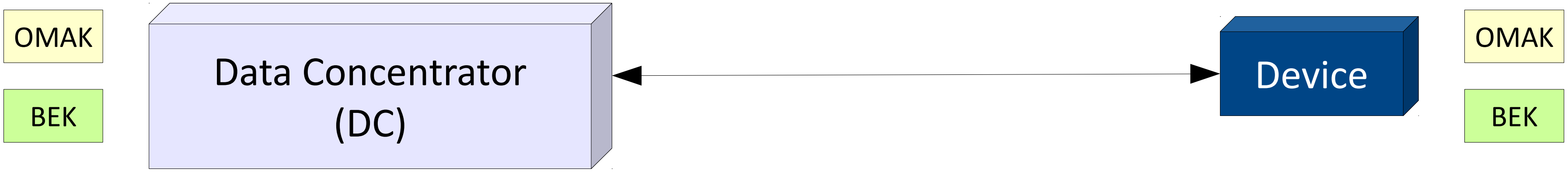
For each message, generate an RC4 cipher stream using the secret "Base Encryption Key" (***BEK***). Encryption by xor of plaintext and cipher stream.



Transmit encrypted message and its digest:



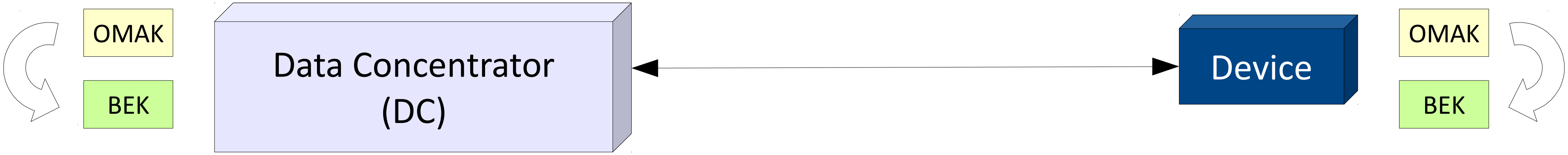
OSGP communication security



OMAK: Data **integrity**
(prevent forging of data)

BEK: Data **confidentiality**
(prevent reading of data)

OSGP communication security

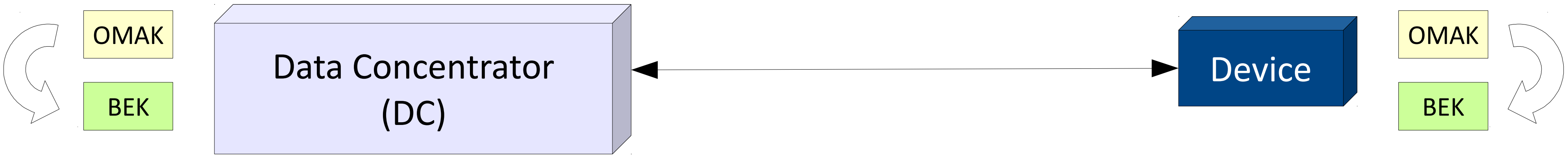


OMAK: Data **integrity**
(prevent forging of data)

BEK: Data **confidentiality**
(prevent reading of data)

BEK is in fact derived from OMAK.

OSGP communication security

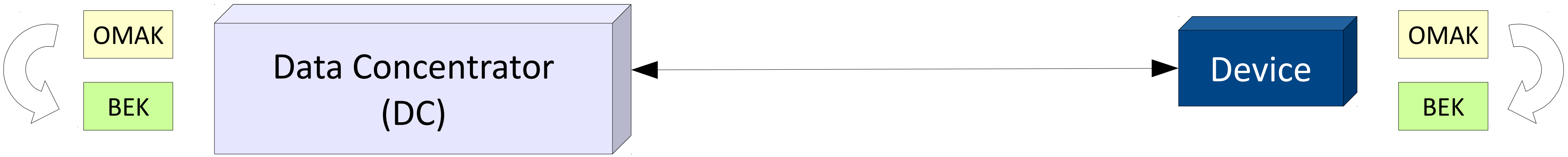


OMAK: Data **integrity**
(prevent forging of data)

BEK: Data **confidentiality**
(prevent reading of data)

BEK is in fact derived from OMAK.
DC and all (!) devices share the same OMAK.

OSGP communication security



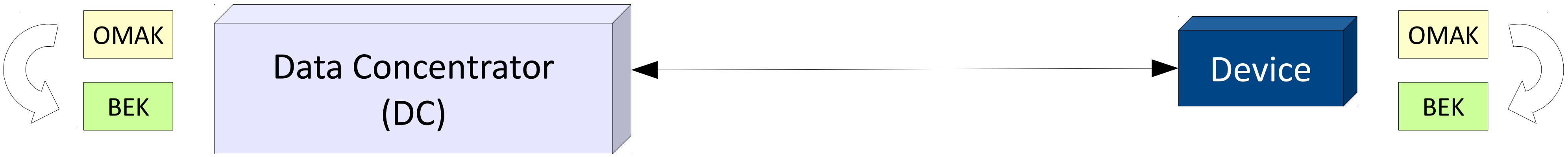
OMAK: Data **integrity**
(prevent forging of data)

BEK: Data **confidentiality**
(prevent reading of data)

BEK is in fact derived from OMAK.
DC and all (!) devices share the same OMAK.

Kursawe and Peters (2015), Jovanovic and Neves (2015) showed how to exploit the OSGP-Digest-Algorithm to derive the OMAK.

OSGP communication security



OMAK: Data **integrity**
(prevent forging of data)

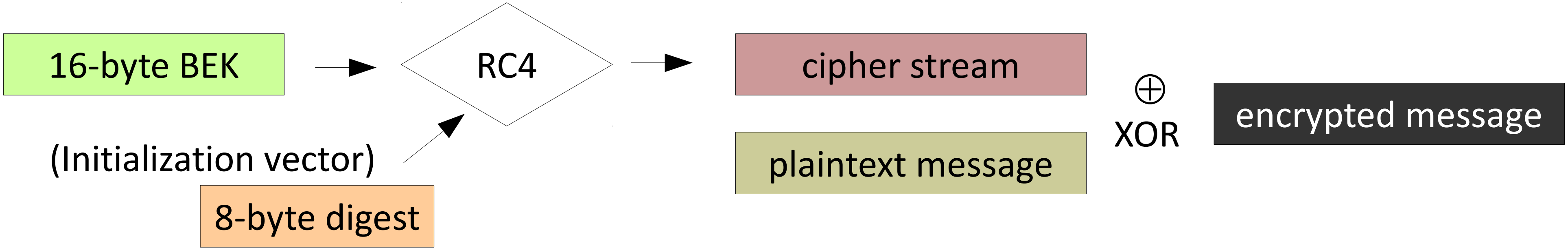
BEK: Data **confidentiality**
(prevent reading of data)

BEK is in fact derived from OMAK.
DC and all (!) devices share the same OMAK.

Kursawe and Peters (2015), Jovanovic and Neves (2015) showed how to exploit the OSGP-Digest-Algorithm to derive the OMAK.

Here, we show an independent attack exploiting RC4 to derive the BEK, thereby compromising OSGP's data confidentiality.

Attack on RC4



16-byte BEK

RC4

(Initialization vector)

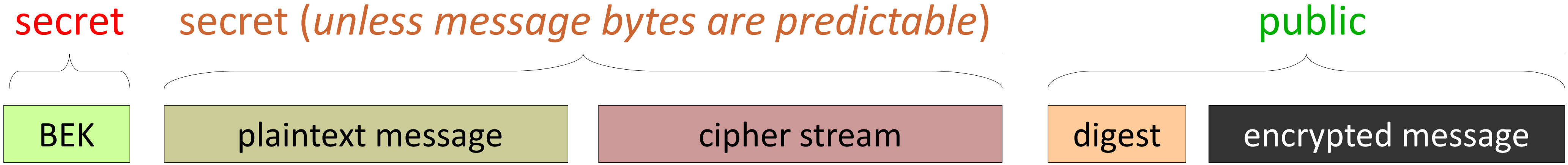
8-byte digest

cipher stream

plaintext message

⊕ XOR

encrypted message



secret

secret (*unless message bytes are predictable*)

public

BEK

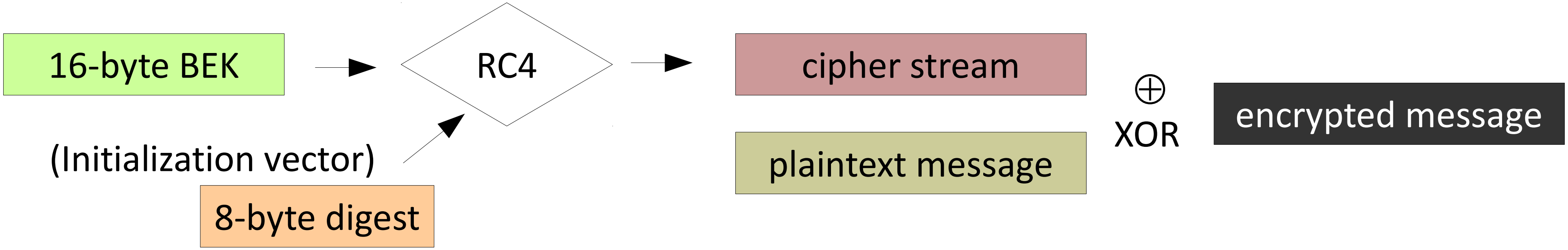
plaintext message

cipher stream

digest

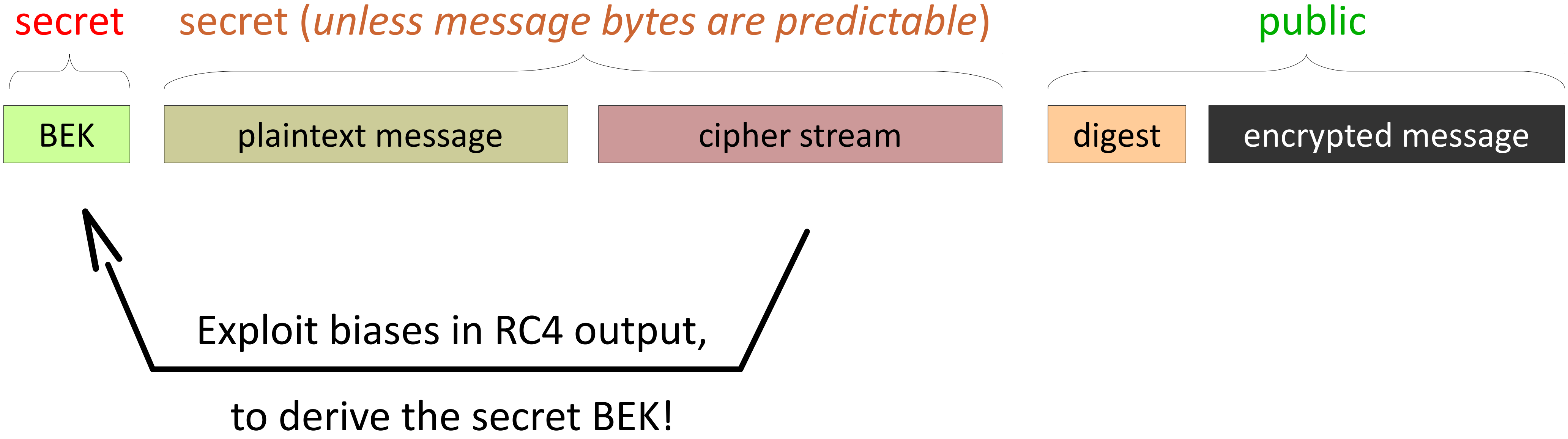
encrypted message

Attack on RC4


16-byte BEK → RC4 → cipher stream

(Initialization vector) 8-byte digest ↗ RC4

cipher stream ⊕ XOR plaintext message — encrypted message


secret — BEK

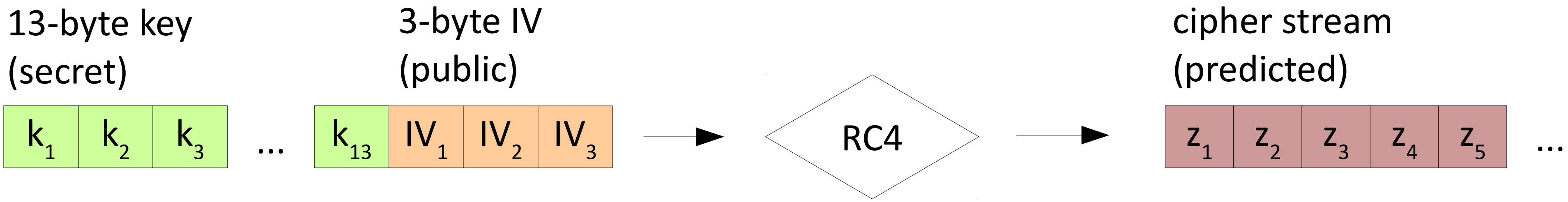
secret (*unless message bytes are predictable*) — plaintext message, cipher stream

public — digest, encrypted message

Exploit biases in RC4 output,

to derive the secret BEK!

Biased output of "classical" RC4



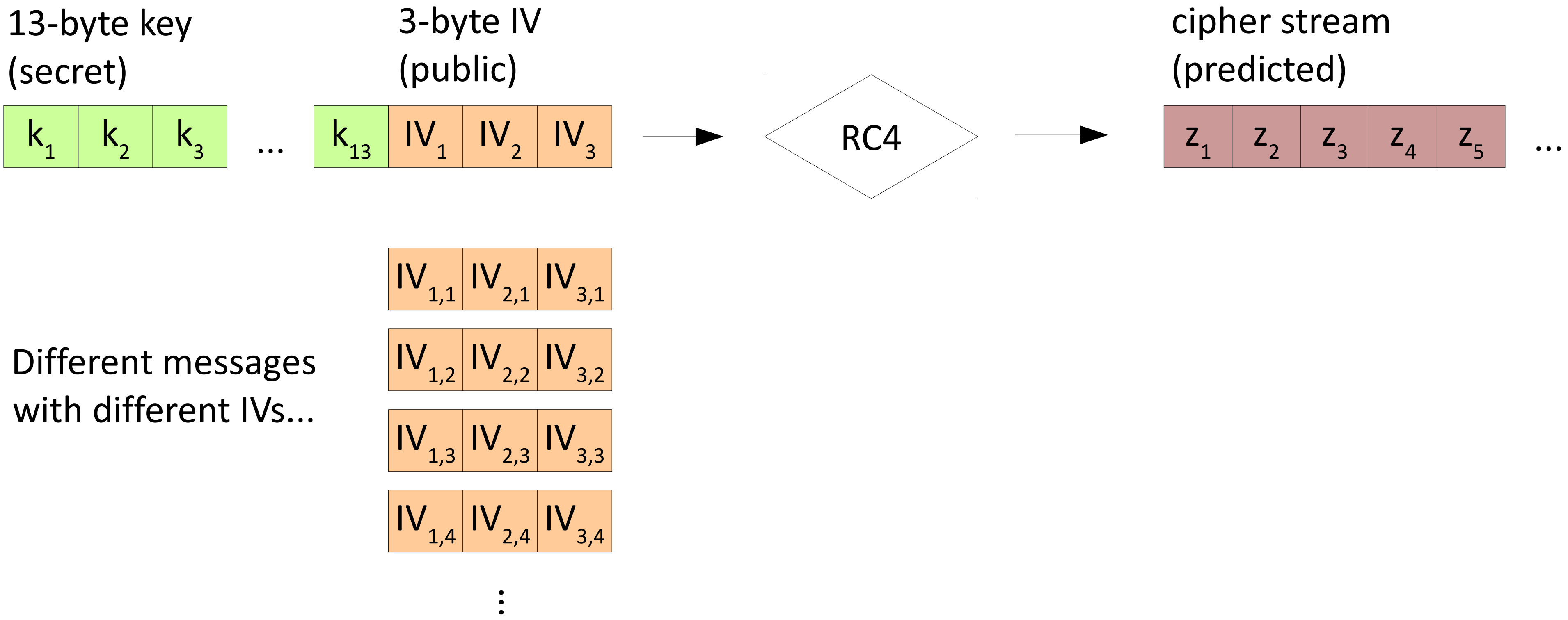
13-byte key
(secret)

3-byte IV
(public)

cipher stream
(predicted)

k_1 k_2 k_3 … k_{13} IV_1 IV_2 IV_3 → RC4 → z_1 z_2 z_3 z_4 z_5 …

Biased output of “classical” RC4



13-byte key
(secret)

k_1 k_2 k_3 … k_{13}

3-byte IV
(public)

IV_1 IV_2 IV_3

RC4

cipher stream
(predicted)

z_1 z_2 z_3 z_4 z_5 …

Different messages
with different IVs...

$IV_{1,1}$ $IV_{2,1}$ $IV_{3,1}$

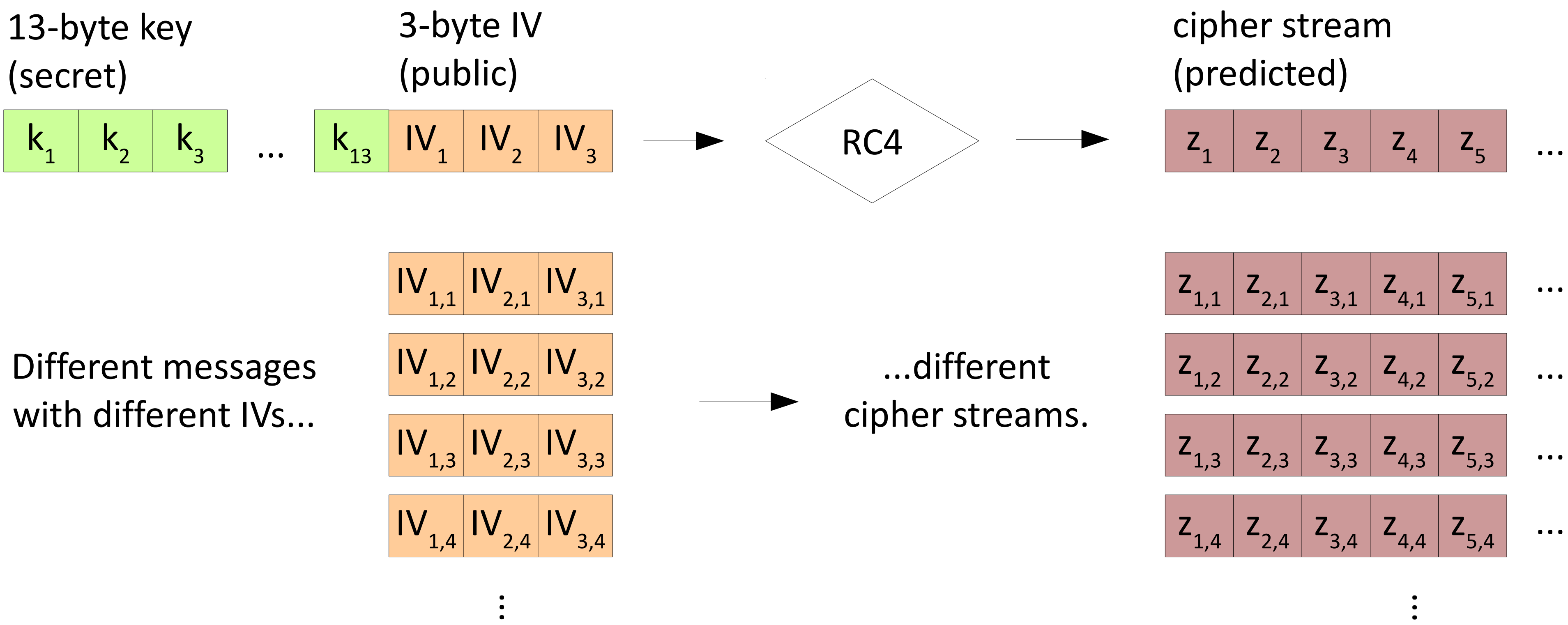
$IV_{1,2}$ $IV_{2,2}$ $IV_{3,2}$

$IV_{1,3}$ $IV_{2,3}$ $IV_{3,3}$

$IV_{1,4}$ $IV_{2,4}$ $IV_{3,4}$

⋮

Biased output of “classical” RC4



13-byte key (secret)

k_1 k_2 k_3 … k_{13}

3-byte IV (public)

IV_1 IV_2 IV_3

RC4

cipher stream (predicted)

z_1 z_2 z_3 z_4 z_5 …

Different messages with different IVs...

$IV_{1,1}$ $IV_{2,1}$ $IV_{3,1}$

$IV_{1,2}$ $IV_{2,2}$ $IV_{3,2}$

$IV_{1,3}$ $IV_{2,3}$ $IV_{3,3}$

$IV_{1,4}$ $IV_{2,4}$ $IV_{3,4}$

⋮

...different cipher streams.

$z_{1,1}$ $z_{2,1}$ $z_{3,1}$ $z_{4,1}$ $z_{5,1}$ …

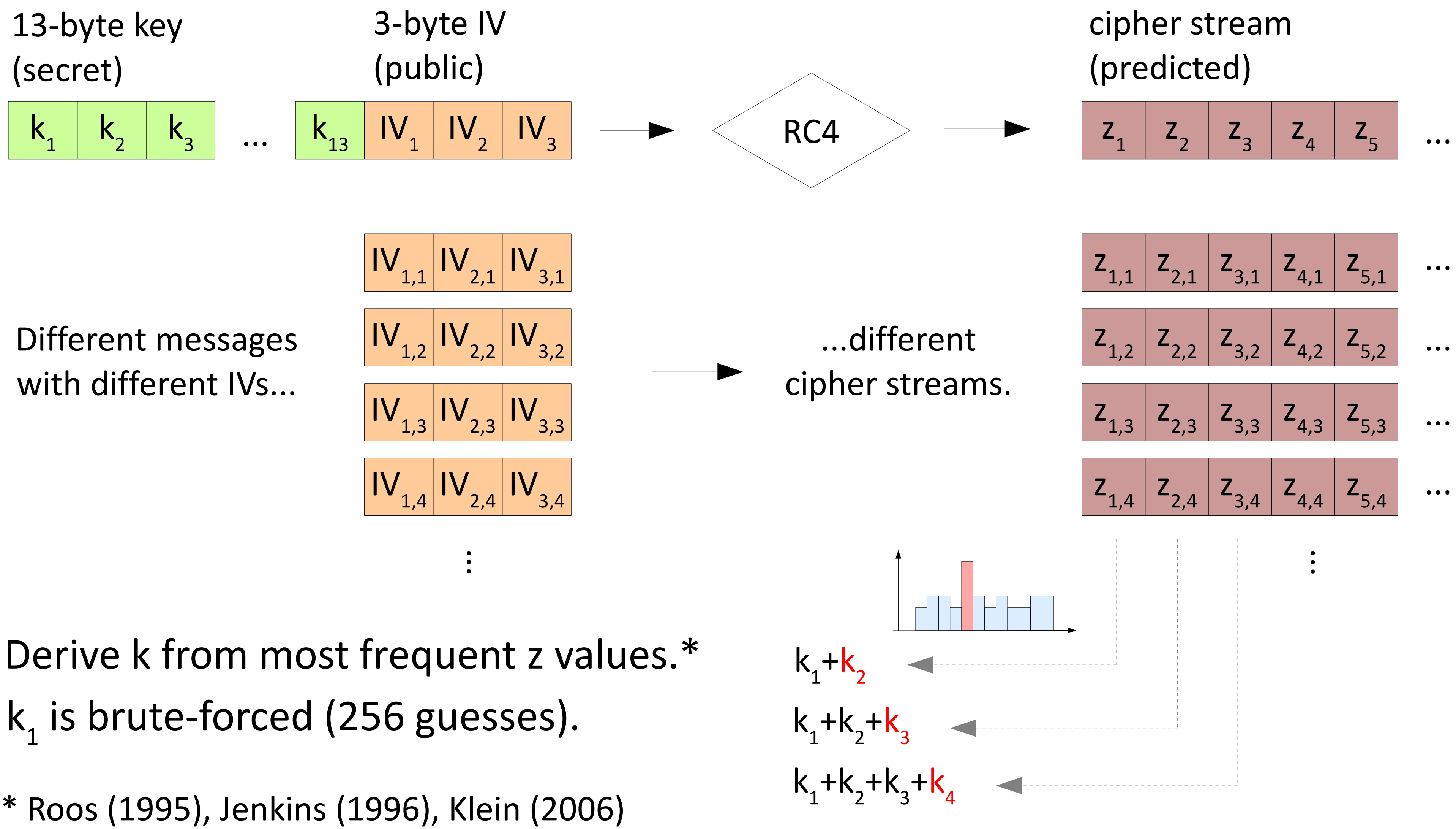
$z_{1,2}$ $z_{2,2}$ $z_{3,2}$ $z_{4,2}$ $z_{5,2}$ …

$z_{1,3}$ $z_{2,3}$ $z_{3,3}$ $z_{4,3}$ $z_{5,3}$ …

$z_{1,4}$ $z_{2,4}$ $z_{3,4}$ $z_{4,4}$ $z_{5,4}$ …

⋮

Biased output of "classical" RC4



13-byte key (secret)

3-byte IV (public)

cipher stream (predicted)

k_1 k_2 k_3 … k_{13} IV_1 IV_2 IV_3 → RC4 → z_1 z_2 z_3 z_4 z_5 …

Different messages with different IVs...

$IV_{1,1}$	$IV_{2,1}$	$IV_{3,1}$
$IV_{1,2}$	$IV_{2,2}$	$IV_{3,2}$
$IV_{1,3}$	$IV_{2,3}$	$IV_{3,3}$
$IV_{1,4}$	$IV_{2,4}$	$IV_{3,4}$
⋮		

...different cipher streams.

$z_{1,1}$	$z_{2,1}$	$z_{3,1}$	$z_{4,1}$	$z_{5,1}$	…
$z_{1,2}$	$z_{2,2}$	$z_{3,2}$	$z_{4,2}$	$z_{5,2}$	…
$z_{1,3}$	$z_{2,3}$	$z_{3,3}$	$z_{4,3}$	$z_{5,3}$	…
$z_{1,4}$	$z_{2,4}$	$z_{3,4}$	$z_{4,4}$	$z_{5,4}$	…
		⋮			

k_1+k_2

$k_1+k_2+k_3$

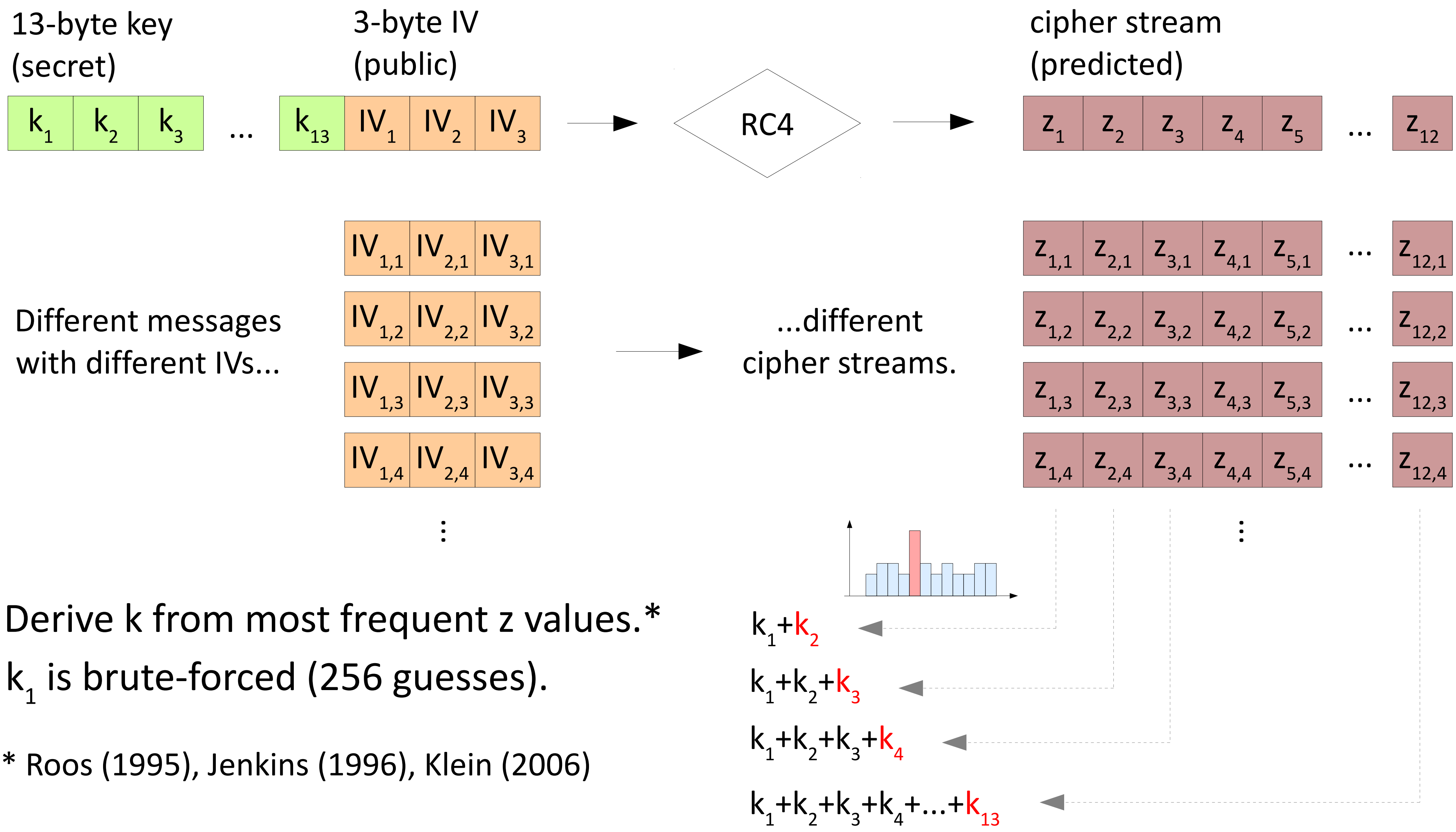
$k_1+k_2+k_3+k_4$

Derive k from most frequent z values.*

k_1 is brute-forced (256 guesses).

* Roos (1995), Jenkins (1996), Klein (2006)

Biased output of “classical” RC4



13-byte key (secret)

3-byte IV (public)

cipher stream (predicted)

k_1 k_2 k_3 … k_{13} IV_1 IV_2 IV_3 → RC4 → z_1 z_2 z_3 z_4 z_5 … z_{12}

Different messages with different IVs...

$IV_{1,1}$ $IV_{2,1}$ $IV_{3,1}$

$IV_{1,2}$ $IV_{2,2}$ $IV_{3,2}$

$IV_{1,3}$ $IV_{2,3}$ $IV_{3,3}$

$IV_{1,4}$ $IV_{2,4}$ $IV_{3,4}$

⋮

→ ...different cipher streams.

$z_{1,1}$ $z_{2,1}$ $z_{3,1}$ $z_{4,1}$ $z_{5,1}$ … $z_{12,1}$

$z_{1,2}$ $z_{2,2}$ $z_{3,2}$ $z_{4,2}$ $z_{5,2}$ … $z_{12,2}$

$z_{1,3}$ $z_{2,3}$ $z_{3,3}$ $z_{4,3}$ $z_{5,3}$ … $z_{12,3}$

$z_{1,4}$ $z_{2,4}$ $z_{3,4}$ $z_{4,4}$ $z_{5,4}$ … $z_{12,4}$

⋮

k_1+k_2

$k_1+k_2+k_3$

$k_1+k_2+k_3+k_4$

$k_1+k_2+k_3+k_4+\ldots+k_{13}$

Derive k from most frequent z values.*

k_1 is brute-forced (256 guesses).

* Roos (1995), Jenkins (1996), Klein (2006)

Biased output of “classical” RC4

Classical RC4

Go through n recorded cipher streams *once* and count most frequent value for |k|-1 cypher bytes. For all 256 values of k_1, calculate and test the key.

Complexity:

O(n*(|k|-1) + 256)

Classical RC4 vs. OSGP RC4

Classical RC4

13-byte key

k_1	k_2	k_3	…	k_{13}

3-byte initialisation vector

IV_1	IV_2	IV_3

Used key (concatenation):

k_1	k_2	k_3	…	k_{13}	IV_1	IV_2	IV_3

Broken! (e.g. WEP)

Roos (1995),
Jenkins (1996),
Klein (2006)

Classical RC4 vs. OSGP RC4

OSGP RC4

16-byte key (BEK)

k_1	k_2	k_3	…	k_{16}

8-byte initialisation vector (message digest)

IV_1	IV_2	IV_3	…	IV_8

Used key (xor of first 8 bytes):

$k_1 \oplus IV_1$	$k_2 \oplus IV_2$	$k_3 \oplus IV_3$	…	$k_8 \oplus IV_8$	k_9	k_{10}	…	k_{16}

Classical RC4

13-byte key

k_1	k_2	k_3	…	k_{13}

3-byte initialisation vector

IV_1	IV_2	IV_3

Used key (concatenation):

k_1	k_2	k_3	…	k_{13}	IV_1	IV_2	IV_3

Broken! (e.g. WEP)

Roos (1995),
Jenkins (1996),
Klein (2006)

Classical RC4 vs. OSGP RC4

OSGP RC4

16-byte key (BEK)

k_1	k_2	k_3	…	k_{16}

8-byte initialisation vector (message digest)

IV_1	IV_2	IV_3	…	IV_8

Used key (xor of first 8 bytes):

$k_1 \oplus IV_1$	$k_2 \oplus IV_2$	$k_3 \oplus IV_3$	…	$k_8 \oplus IV_8$	k_9	k_{10}	…	k_{16}

Broken?

Classical RC4

13-byte key

k_1	k_2	k_3	…	k_{13}

3-byte initialisation vector

IV_1	IV_2	IV_3

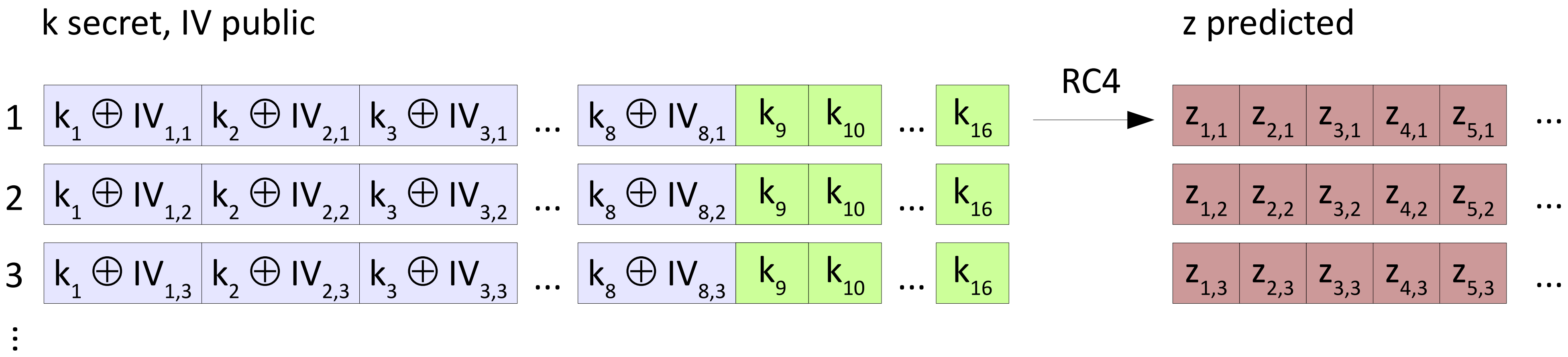
Used key (concatenation):

k_1	k_2	k_3	…	k_{13}	IV_1	IV_2	IV_3

Broken! (e.g. WEP)

Roos (1995),
Jenkins (1996),
Klein (2006)

Attack on OSGP RC4

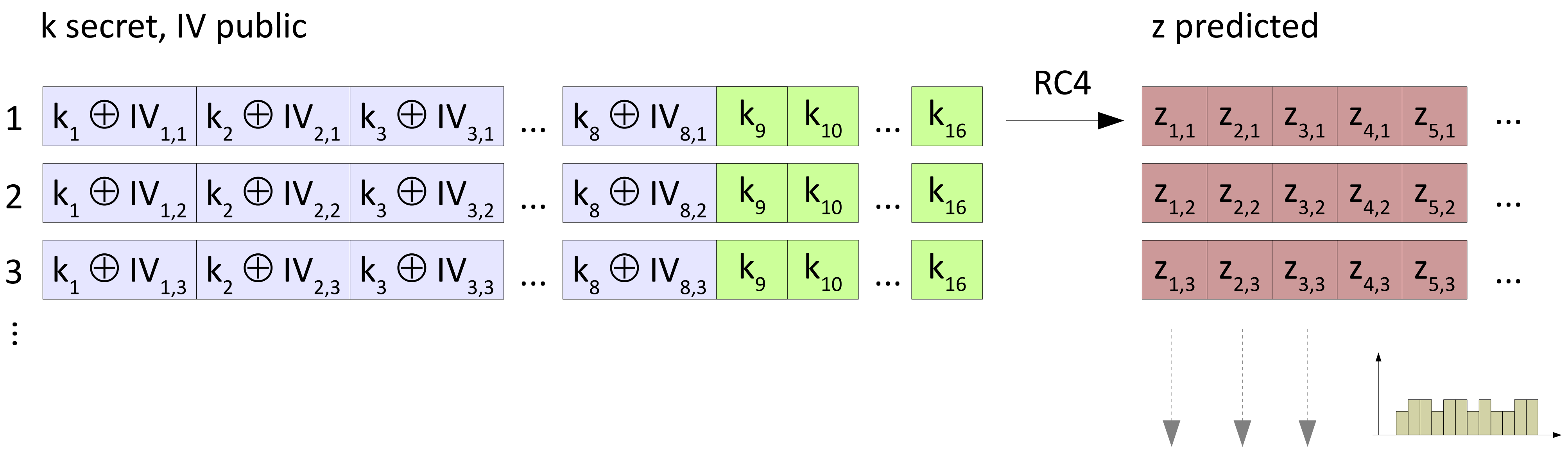


k secret, IV public

z predicted

1	$k_1 \oplus IV_{1,1}$	$k_2 \oplus IV_{2,1}$	$k_3 \oplus IV_{3,1}$	…	$k_8 \oplus IV_{8,1}$	k_9	k_{10}	…	k_{16}	$\xrightarrow{\text{RC4}}$	$z_{1,1}$	$z_{2,1}$	$z_{3,1}$	$z_{4,1}$	$z_{5,1}$	…
2	$k_1 \oplus IV_{1,2}$	$k_2 \oplus IV_{2,2}$	$k_3 \oplus IV_{3,2}$	…	$k_8 \oplus IV_{8,2}$	k_9	k_{10}	…	k_{16}		$z_{1,2}$	$z_{2,2}$	$z_{3,2}$	$z_{4,2}$	$z_{5,2}$	…
3	$k_1 \oplus IV_{1,3}$	$k_2 \oplus IV_{2,3}$	$k_3 \oplus IV_{3,3}$	…	$k_8 \oplus IV_{8,3}$	k_9	k_{10}	…	k_{16}		$z_{1,3}$	$z_{2,3}$	$z_{3,3}$	$z_{4,3}$	$z_{5,3}$	…
⋮																

Attack on OSGP RC4



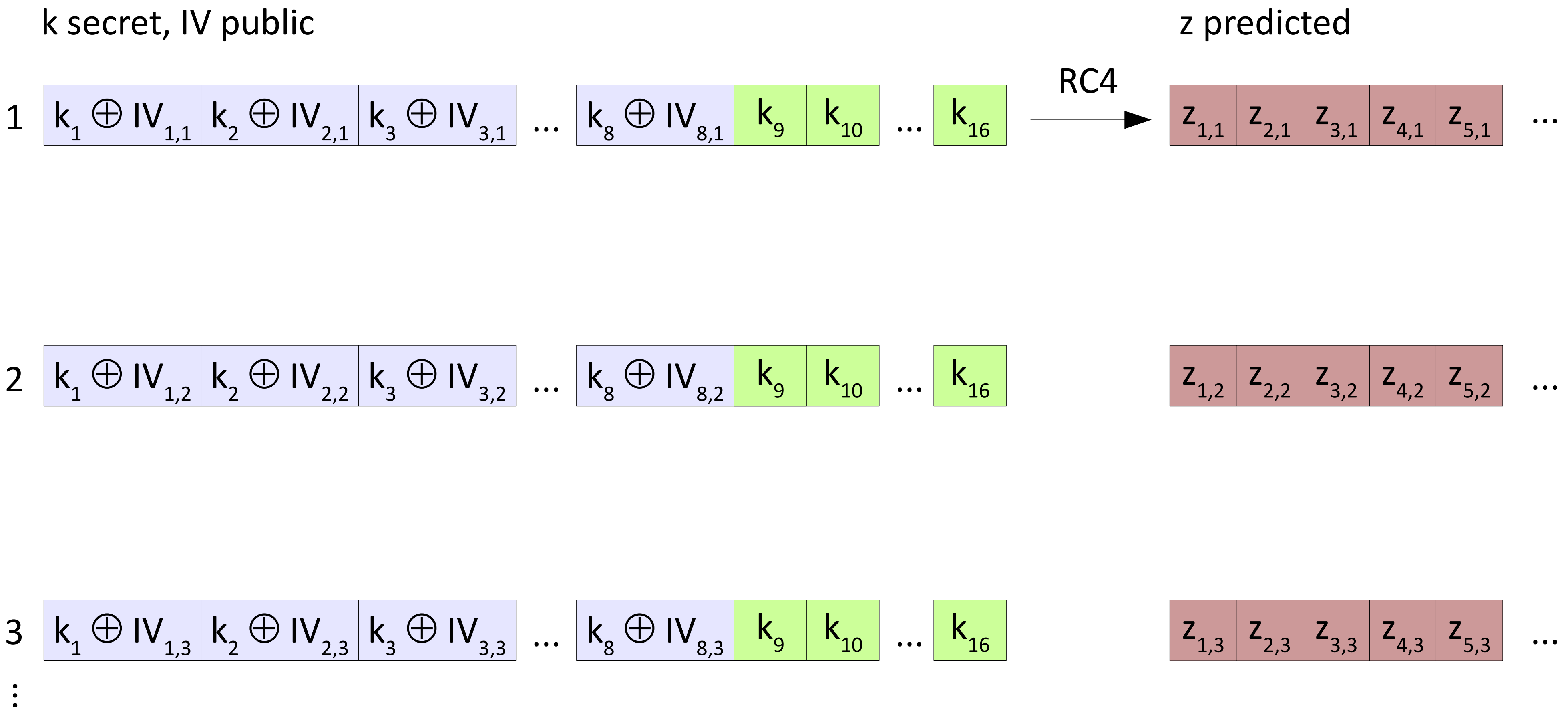
k secret, IV public

z predicted

1	$k_1 \oplus IV_{1,1}$	$k_2 \oplus IV_{2,1}$	$k_3 \oplus IV_{3,1}$	…	$k_8 \oplus IV_{8,1}$	k_9	k_{10}	…	k_{16}	RC4 →	$z_{1,1}$	$z_{2,1}$	$z_{3,1}$	$z_{4,1}$	$z_{5,1}$	…
2	$k_1 \oplus IV_{1,2}$	$k_2 \oplus IV_{2,2}$	$k_3 \oplus IV_{3,2}$	…	$k_8 \oplus IV_{8,2}$	k_9	k_{10}	…	k_{16}		$z_{1,2}$	$z_{2,2}$	$z_{3,2}$	$z_{4,2}$	$z_{5,2}$	…
3	$k_1 \oplus IV_{1,3}$	$k_2 \oplus IV_{2,3}$	$k_3 \oplus IV_{3,3}$	…	$k_8 \oplus IV_{8,3}$	k_9	k_{10}	…	k_{16}		$z_{1,3}$	$z_{2,3}$	$z_{3,3}$	$z_{4,3}$	$z_{5,3}$	…
⋮																

Counting most frequent values is distorted by different IVs for each message.

Biased output of “classical” RC4

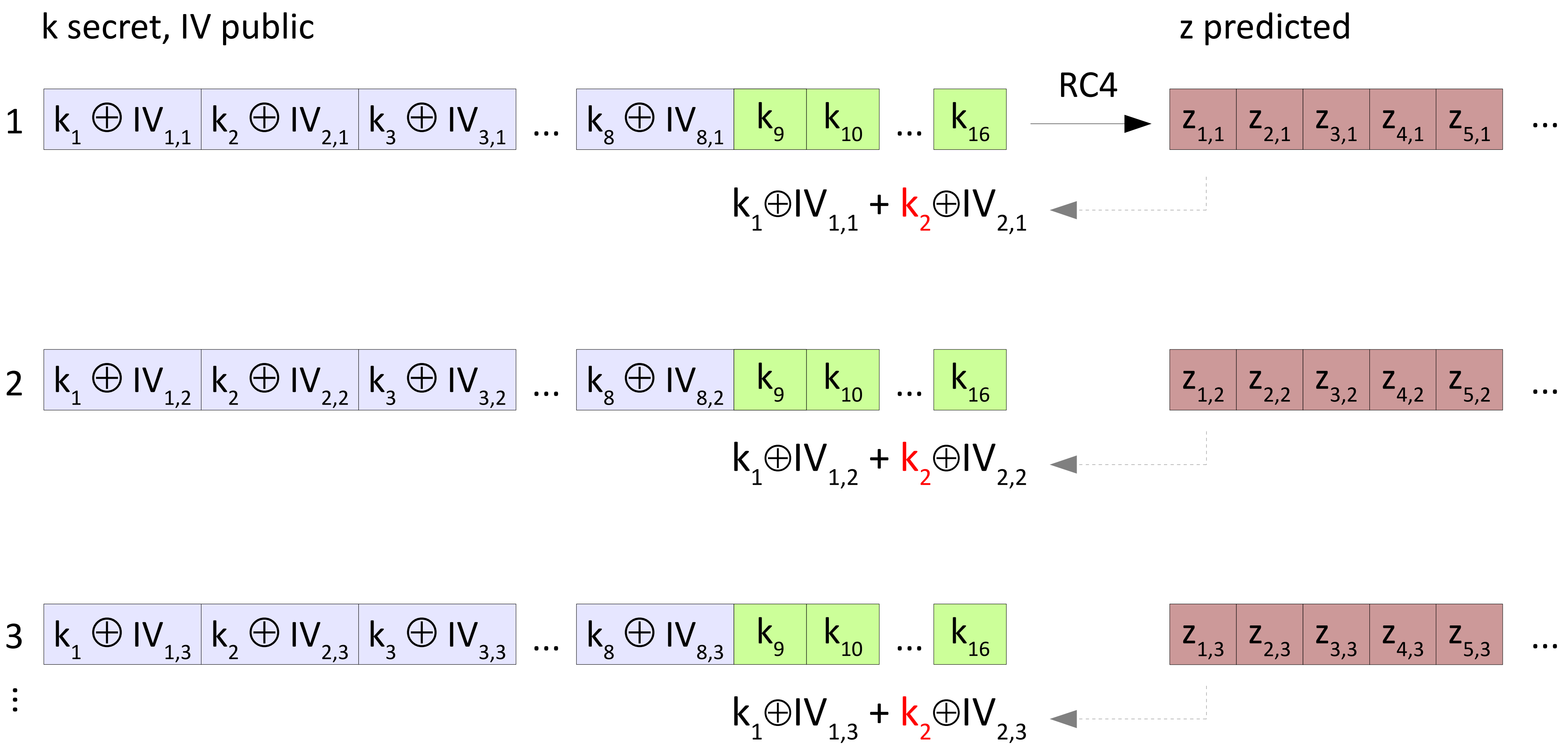


k secret, IV public

z predicted

1	$k_1 \oplus IV_{1,1}$	$k_2 \oplus IV_{2,1}$	$k_3 \oplus IV_{3,1}$	…	$k_8 \oplus IV_{8,1}$	k_9	k_{10}	…	k_{16}	$\xrightarrow{RC4}$	$z_{1,1}$	$z_{2,1}$	$z_{3,1}$	$z_{4,1}$	$z_{5,1}$	…
2	$k_1 \oplus IV_{1,2}$	$k_2 \oplus IV_{2,2}$	$k_3 \oplus IV_{3,2}$	…	$k_8 \oplus IV_{8,2}$	k_9	k_{10}	…	k_{16}		$z_{1,2}$	$z_{2,2}$	$z_{3,2}$	$z_{4,2}$	$z_{5,2}$	…
3	$k_1 \oplus IV_{1,3}$	$k_2 \oplus IV_{2,3}$	$k_3 \oplus IV_{3,3}$	…	$k_8 \oplus IV_{8,3}$	k_9	k_{10}	…	k_{16}		$z_{1,3}$	$z_{2,3}$	$z_{3,3}$	$z_{4,3}$	$z_{5,3}$	…
⋮																

Biased output of “classical” RC4



k secret, IV public

z predicted

1 $k_1 \oplus IV_{1,1}$ $k_2 \oplus IV_{2,1}$ $k_3 \oplus IV_{3,1}$ … $k_8 \oplus IV_{8,1}$ k_9 k_{10} … k_{16} $\xrightarrow{\text{RC4}}$ $z_{1,1}$ $z_{2,1}$ $z_{3,1}$ $z_{4,1}$ $z_{5,1}$ …

$k_1 \oplus IV_{1,1} + k_2 \oplus IV_{2,1}$

2 $k_1 \oplus IV_{1,2}$ $k_2 \oplus IV_{2,2}$ $k_3 \oplus IV_{3,2}$ … $k_8 \oplus IV_{8,2}$ k_9 k_{10} … k_{16} $z_{1,2}$ $z_{2,2}$ $z_{3,2}$ $z_{4,2}$ $z_{5,2}$ …

$k_1 \oplus IV_{1,2} + k_2 \oplus IV_{2,2}$

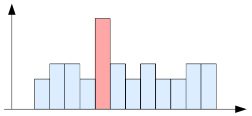
3 $k_1 \oplus IV_{1,3}$ $k_2 \oplus IV_{2,3}$ $k_3 \oplus IV_{3,3}$ … $k_8 \oplus IV_{8,3}$ k_9 k_{10} … k_{16} $z_{1,3}$ $z_{2,3}$ $z_{3,3}$ $z_{4,3}$ $z_{5,3}$ …

⋮

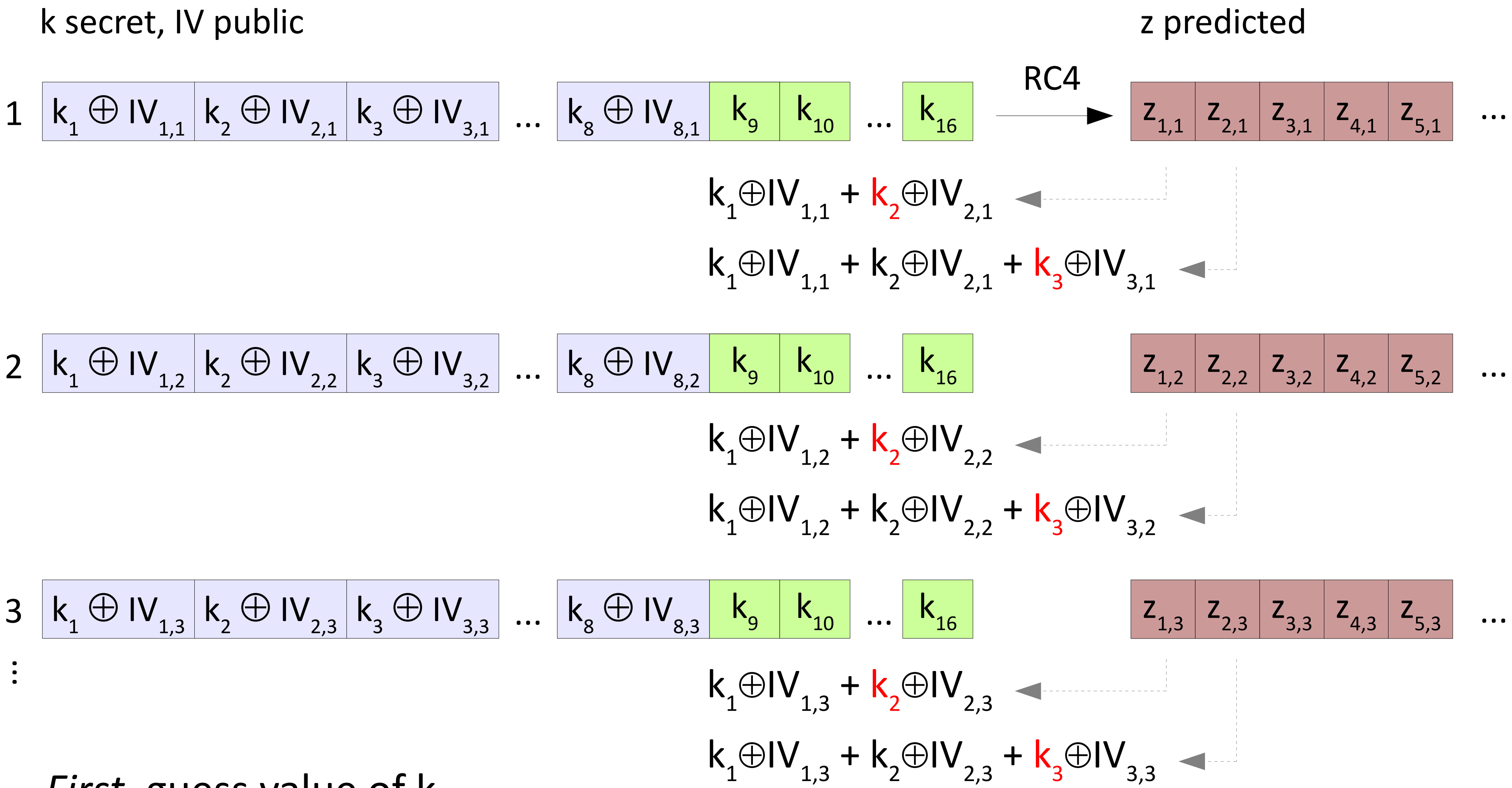
$k_1 \oplus IV_{1,3} + k_2 \oplus IV_{2,3}$

First, guess value of k_1.

Then derive most frequent k values one after the other.



Biased output of “classical” RC4



k secret, IV public

z predicted

1 $k_1 \oplus IV_{1,1}$ | $k_2 \oplus IV_{2,1}$ | $k_3 \oplus IV_{3,1}$ … $k_8 \oplus IV_{8,1}$ | k_9 | k_{10} … k_{16} $\xrightarrow{\text{RC4}}$ $z_{1,1}$ | $z_{2,1}$ | $z_{3,1}$ | $z_{4,1}$ | $z_{5,1}$ …

$k_1 \oplus IV_{1,1} + k_2 \oplus IV_{2,1}$

$k_1 \oplus IV_{1,1} + k_2 \oplus IV_{2,1} + k_3 \oplus IV_{3,1}$

2 $k_1 \oplus IV_{1,2}$ | $k_2 \oplus IV_{2,2}$ | $k_3 \oplus IV_{3,2}$ … $k_8 \oplus IV_{8,2}$ | k_9 | k_{10} … k_{16} $z_{1,2}$ | $z_{2,2}$ | $z_{3,2}$ | $z_{4,2}$ | $z_{5,2}$ …

$k_1 \oplus IV_{1,2} + k_2 \oplus IV_{2,2}$

$k_1 \oplus IV_{1,2} + k_2 \oplus IV_{2,2} + k_3 \oplus IV_{3,2}$

3 $k_1 \oplus IV_{1,3}$ | $k_2 \oplus IV_{2,3}$ | $k_3 \oplus IV_{3,3}$ … $k_8 \oplus IV_{8,3}$ | k_9 | k_{10} … k_{16} $z_{1,3}$ | $z_{2,3}$ | $z_{3,3}$ | $z_{4,3}$ | $z_{5,3}$ …

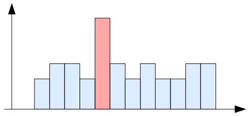
⋮

$k_1 \oplus IV_{1,3} + k_2 \oplus IV_{2,3}$

$k_1 \oplus IV_{1,3} + k_2 \oplus IV_{2,3} + k_3 \oplus IV_{3,3}$

First, guess value of k_1.

Then derive most frequent k values one after the other.



Classical RC4 vs. OSGP RC4

Classical RC4

Go through n recorded cipher streams *once* and count most frequent value for |k|-1 cypher bytes. For all 256 values of k_1, calculate and test the key.

Complexity:

O(n*(|k|-1) + 256)

Classical RC4 vs. OSGP RC4

OSGP RC4

```
For all 256 values of $k_1$:
    Go through n recorded cipher streams
    once for |k|-1 cypher bytes.
    Count most frequent value for each $k_i$.
    Test the key.
```

Complexity:

O(n*(|k|-1)*256)

Classical RC4

Go through n recorded cipher streams *once* and count most frequent value for |k|-1 cypher bytes.

For all 256 values of k_1, calculate and test the key.

Complexity:

O(n*(|k|-1) + 256)

Classical RC4 vs. OSGP RC4

OSGP RC4

For all 256 values of k_1:

- Go through n recorded cipher streams once for |k|-1 cypher bytes.
- Count most frequent value for each k_i.
- Test the key.

Complexity:

O(n*(|k|-1)*256)

Only linear increase of complexity.

Classical RC4

Go through n recorded cipher streams *once* and count most frequent value for |k|-1 cypher bytes.

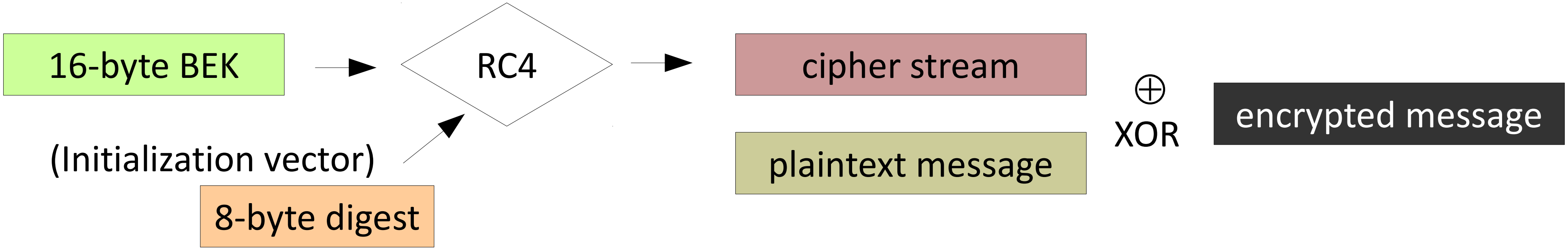
For all 256 values of k_1, calculate and test the key.

Complexity:

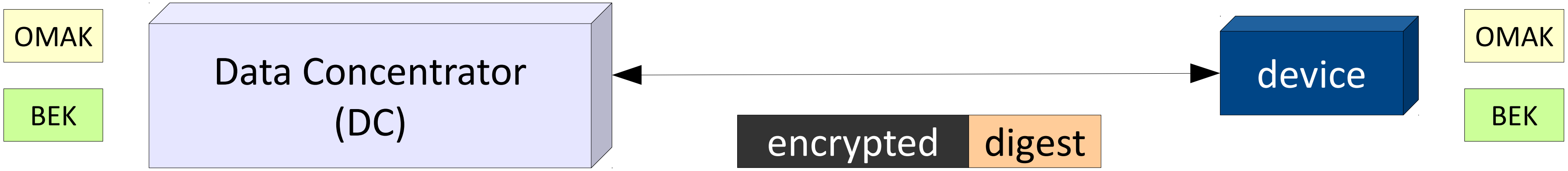
O(n*(|k|-1) + 256)

Practicality of the attack

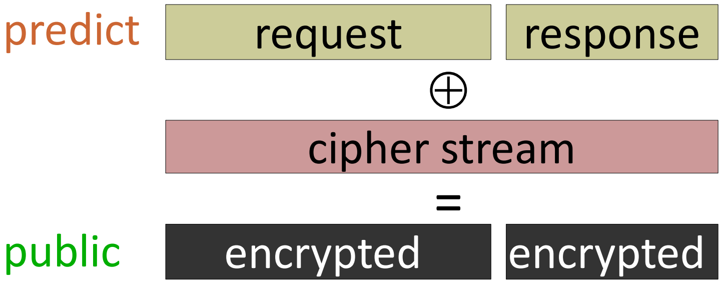
Predicting enough cipher stream bytes.



Transmit encrypted message and its digest:



Practicality of the attack



predict request response

⊕

cipher stream

=

public encrypted encrypted

Predicting enough cipher stream bytes.

E.g. OSGP message to read out device clock.

(Response continues with same cipher stream after request.)

DC request:

1: 0x3F Message ID *(partial table read)*

2: 0x00 Table ID *(device clock)*
3: 0x34

4: 0x00 Table offset
5: 0x00 *(Where to start reading?)*
6: 0x00

7: 0x00 Count
8: 0x06 (How many bytes to read?)

9: 0x?? Message sequence number
10: 0x?? (Individual for each device.
11: 0x?? Hard to predict.)
12: 0x??

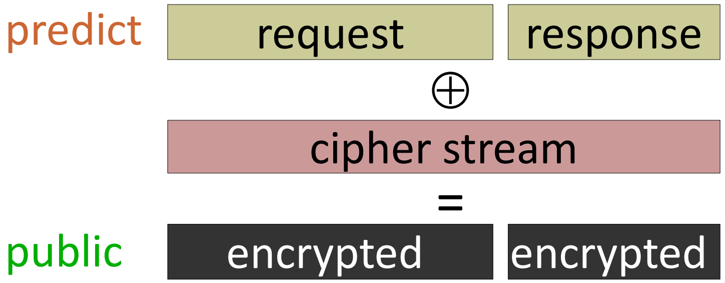
Device response:

13: 0x00 Device answer ("OK")

14: 0x00 Count
15: 0x06 (Same as in request.)

Remaining answer bytes irrelevant, as only 15 bytes are needed.

Practicality of the attack



Predicting enough cipher stream bytes.

E.g. OSGP message to read out device clock.

(Response continues with same cipher stream after request.)

DC request:

1: 0x3F Message ID *(partial table read)*

2: 0x00 Table ID *(device clock)*
3: 0x34

4: 0x00 Table offset
5: 0x00 *(Where to start reading?)*
6: 0x00

7: 0x00 Count
8: 0x06 (How many bytes to read?)

9: 0x?? Message sequence number
10: 0x?? (Individual for each device.
11: 0x?? Hard to predict.)
12: 0x??

Device response:

13: 0x00 Device answer ("OK")

14: 0x00 Count
15: 0x06 (Same as in request.)

Remaining answer bytes irrelevant, as only 15 bytes are needed.

Five bytes must be brute-forced, taking about 2 weeks on a single 3.40 GHz Intel i7-4770.

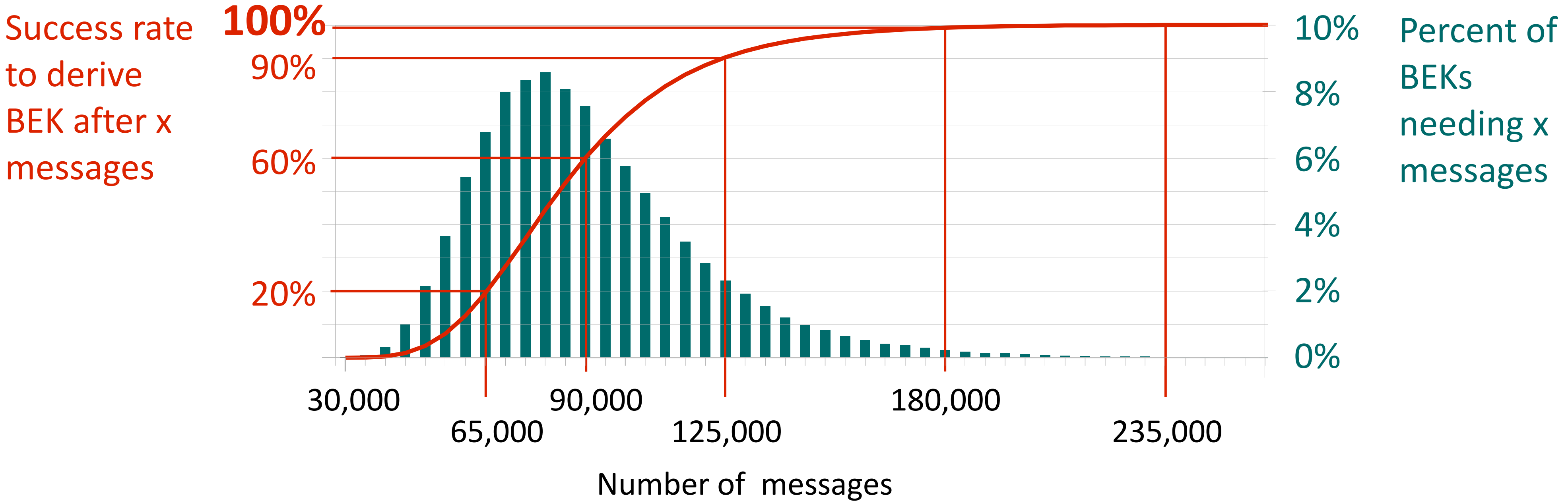
256 cores only take 1.5 hours.

Practicality of the attack

How many eavesdropped messages?

Simulated 145,000 random BEKs, and for each
up to 300,000 messages with random sequence numbers.

Processing 50,000 messages per second on single CPU.

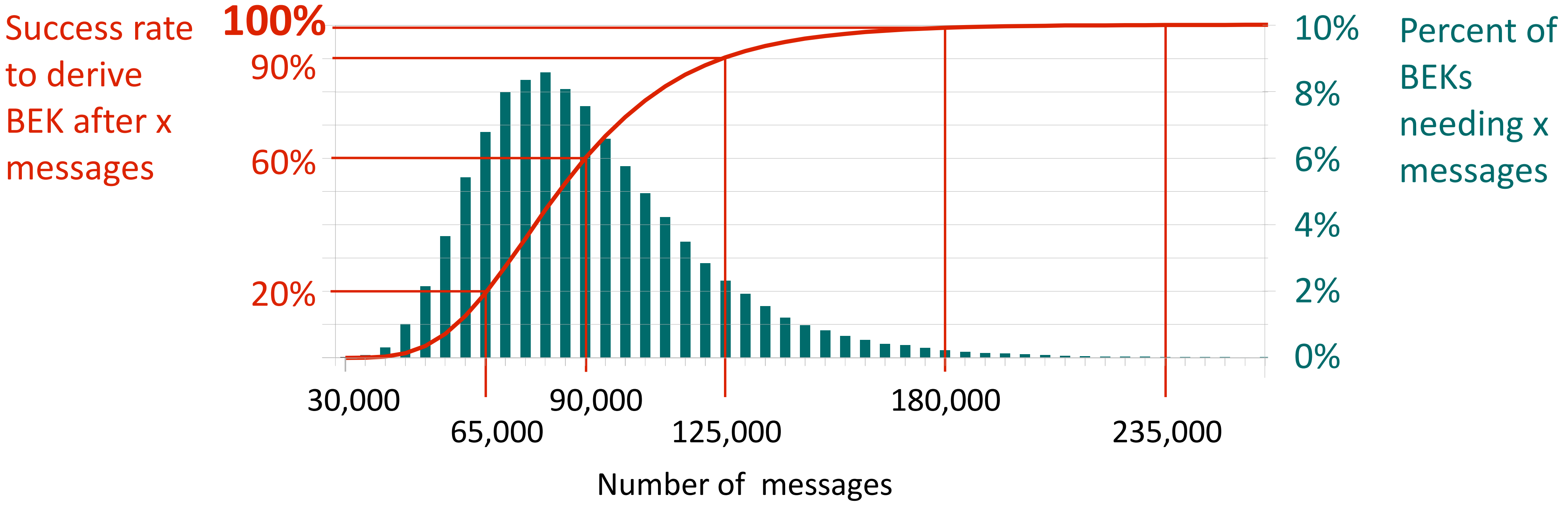


Practicality of the attack

How many eavesdropped messages?

Simulated 145,000 random BEKs, and for each
up to 300,000 messages with random sequence numbers.

Processing 50,000 messages per second on single CPU.

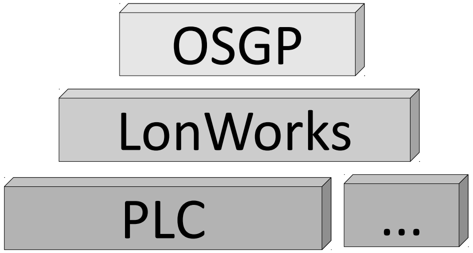


(Attack was refined to derive *all* possible BEKs; simple version only gets 85%.)

Countermeasures?

Do ***not rely*** on OSGP encryption for security!

If possible use encryption on lower layers.

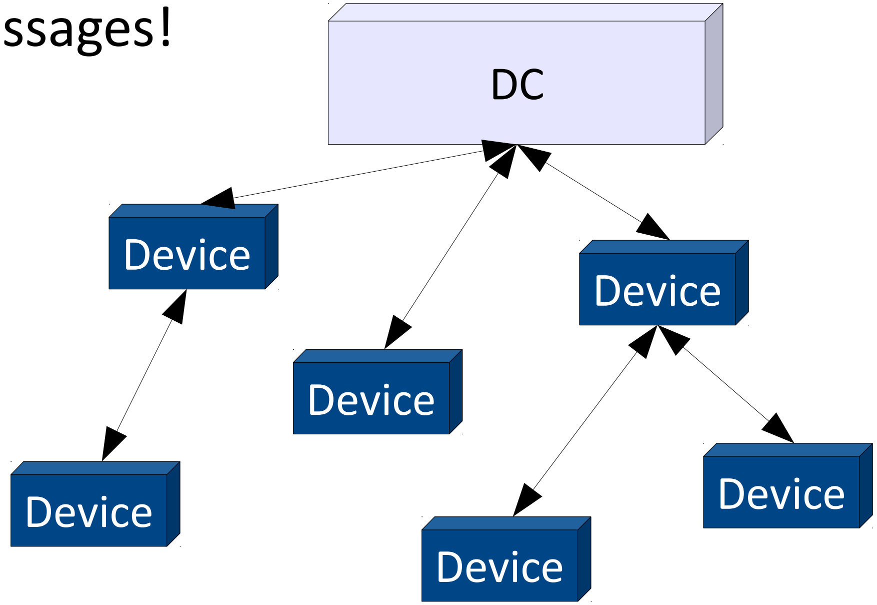


Do ***not*** let an attacker eavesdrop 90,000+ messages!

(1 day = 86,400 seconds)

If possible reduce traffic or

often update the secret OMAK.



The OSGP Alliance has been informed.

They are working on a complete

overhaul of OSGP...

RC4 in detail

```
$S_{-1} = (0, 1, 2, \ldots, 255);$
$i_{-1} = j_{-1} = 0;$
// Shuffle $S$:
for ($r = 0$; $r \leq 255$; $r$++) {
    $i_r = r;$
    $j_r = j_{r-1} + S_{r-1}[i_r] + k[i_r\%16]\ \%\ 256;$
    $S_r[j_r] = S_{r-1}[i_r];$
    $S_r[i_r] = S_{r-1}[j_r];$
}
```

Key scheduling algorithm (KSA)

```
// $S_{-1}$ is the output of KSA.
$i_{-1} = j_{-1} = 0;$
// Generate cipher stream $z$:
for ($r = 0$; ; $r$++) {
    $i_r = r\ \%\ 256;$
    $j_r = j_{r-1} + S_{r-1}[i_r]\ \%\ 256;$
    // Swap $S[j]$ and $S[i]$.
    $S_r[j_r] = S_{r-1}[i_r];$
    $S_r[i_r] = S_{r-1}[j_r];$
    // Output of cipher stream byte.
    $z[r] = S_r[S_r[i_r] + S_r[j_r]\ \%\ 256];$
}
```

Pseudo-Random Generation Algorithm (PRGA)