An evolutionary computation attack on one-round TEA

Eddie YT Ma, MSc
Charlie Obimbo, PhD

University of Guelph, Ontario, Canada
for CAS 2011, Chicago

Agenda

- Problem Overview
  - Tiny Encryption Algorithm
  - Evolutionary Computation
- Experimental Design
- Results
- Conclusions

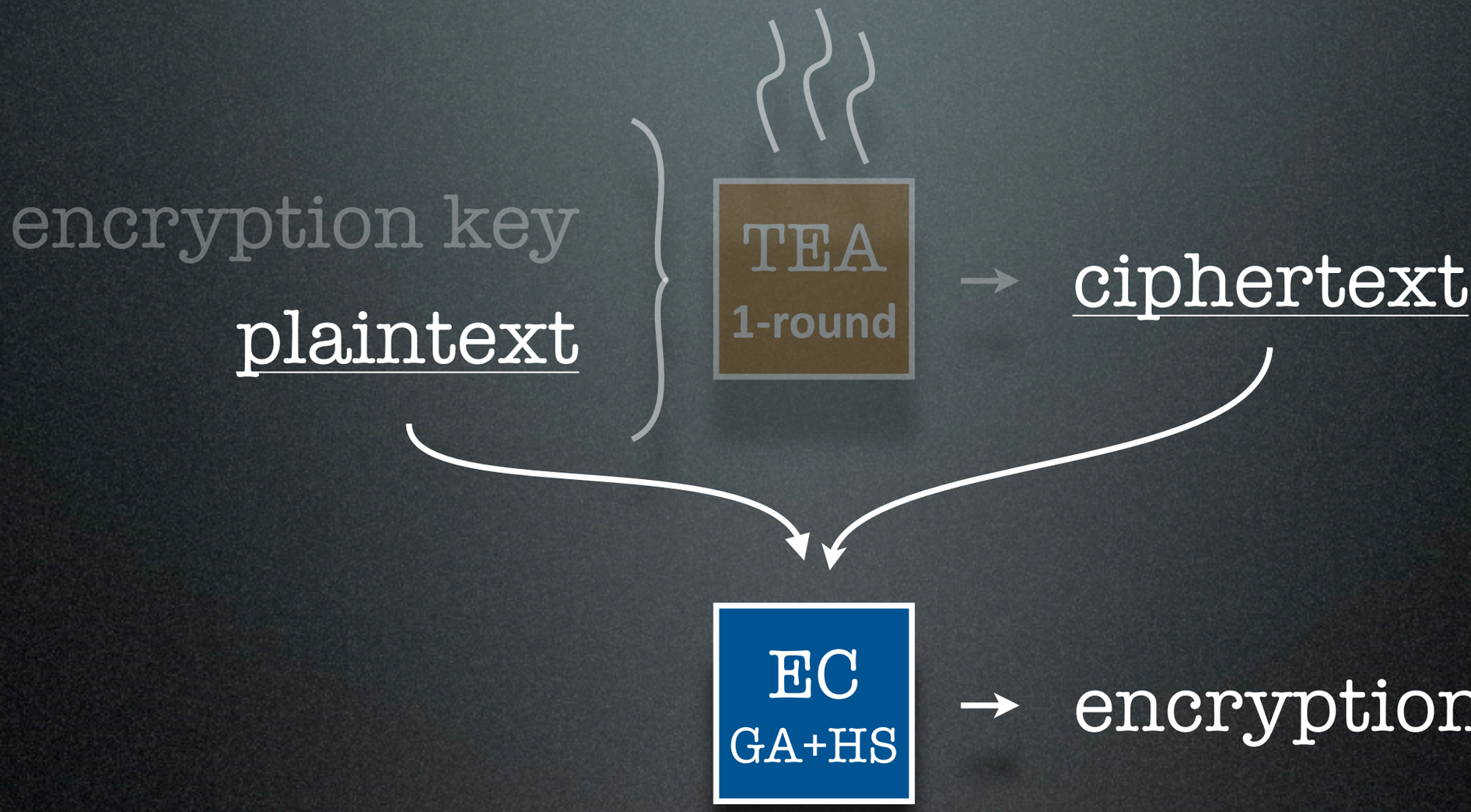
Problem Overview

- In this project we attack one-round TEA with evolutionary computation.

Problem Overview

TEA is an elegant, light weight Feistel block cipher.
(Wheeler & Needham, 1994)

Problem Overview



We use an evolutionary computation algorithm to derive the encryption key.

Problem Overview

- This is called a *known-plaintext* attack.

Agenda

- Problem Overview
  - Tiny Encryption Algorithm ←
  - Evolutionary Computation
- Experimental Design
- Results
- Conclusions

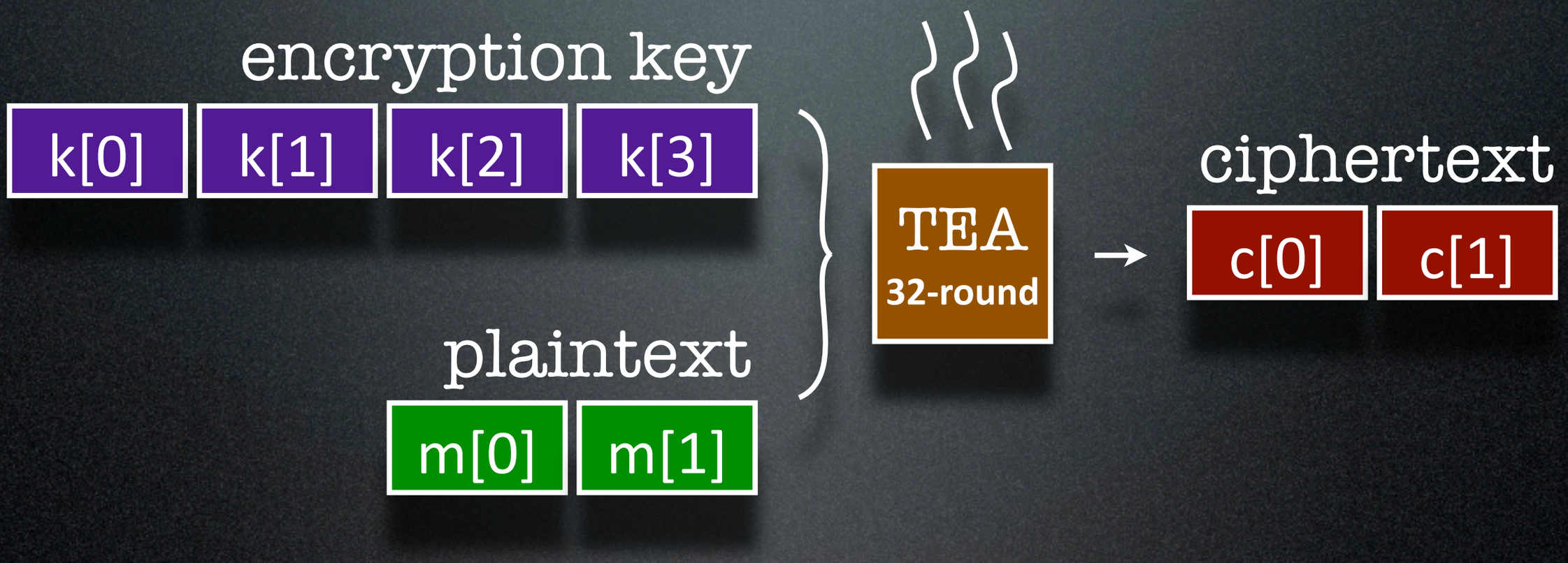
TEA

Tiny Encryption Algorithm

- TEA is a *Feistel block cipher*.
- rounds of operation on the bits of plaintext
- operations are substitutions, permutations
- want: *unique sequence of operations for each key*

TEA

Tiny Encryption Algorithm

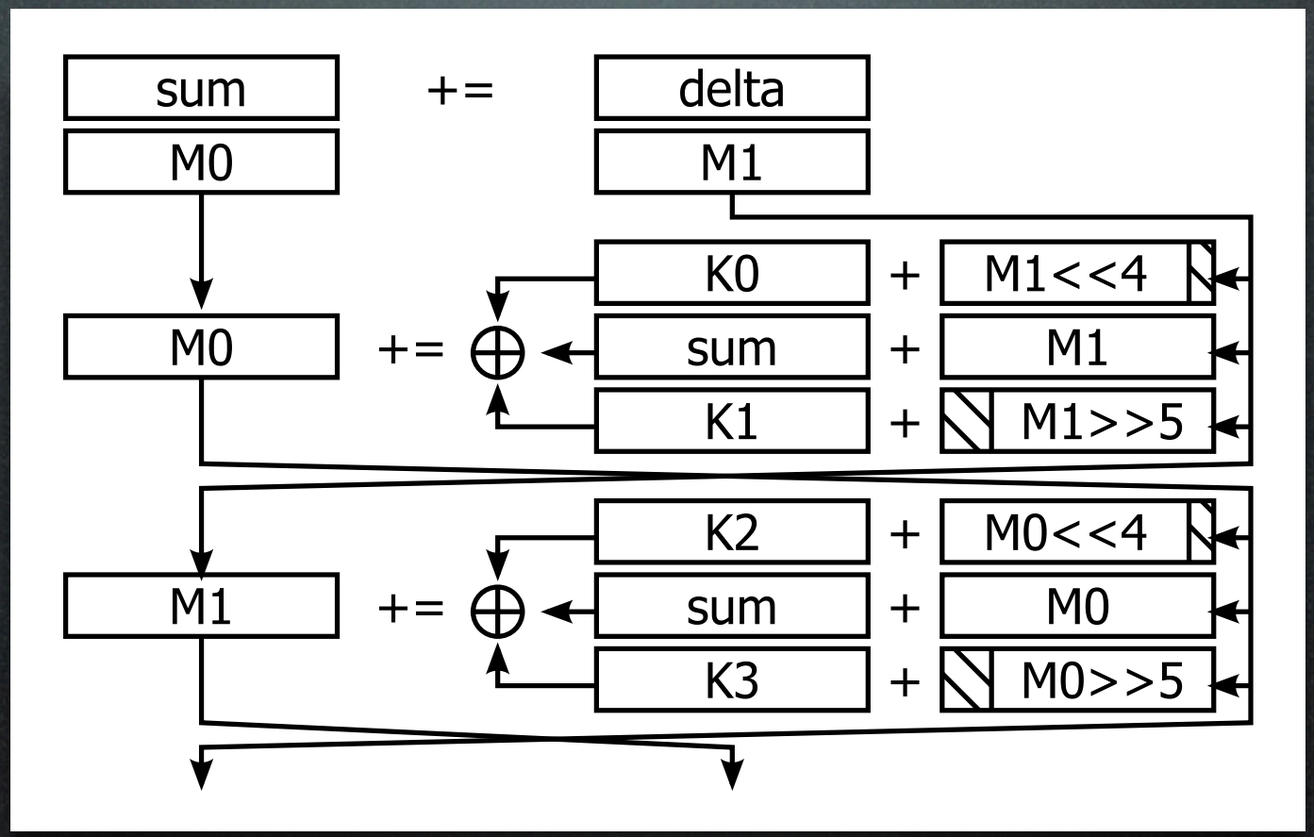


The key is 128-bits long (4 × 32-bit words)

Text is operated in blocks of 64-bits (2 × 32-bit words)

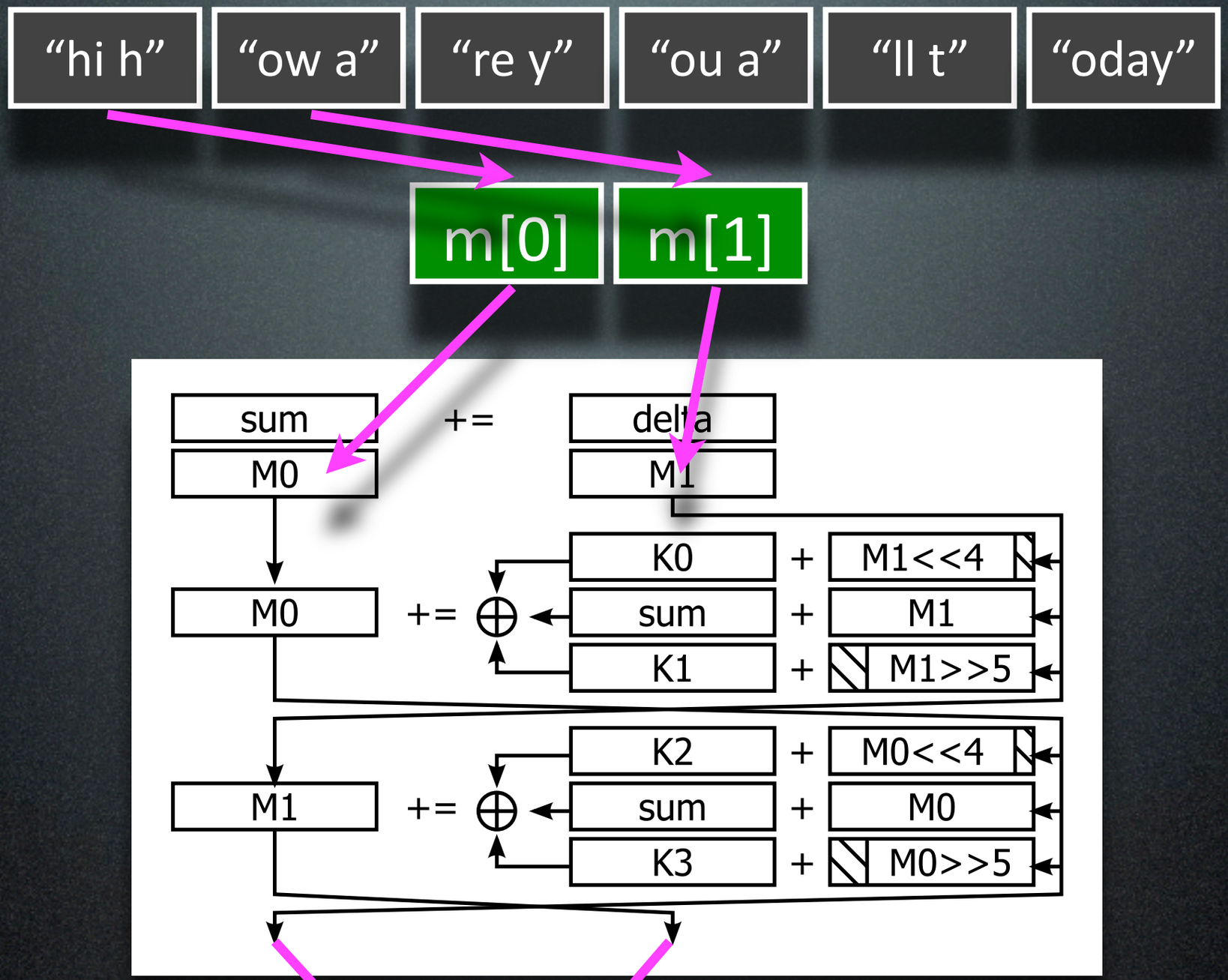
TEA

Tiny Encryption Algorithm



`delta` is a constant

`sum` is as a key scheduler

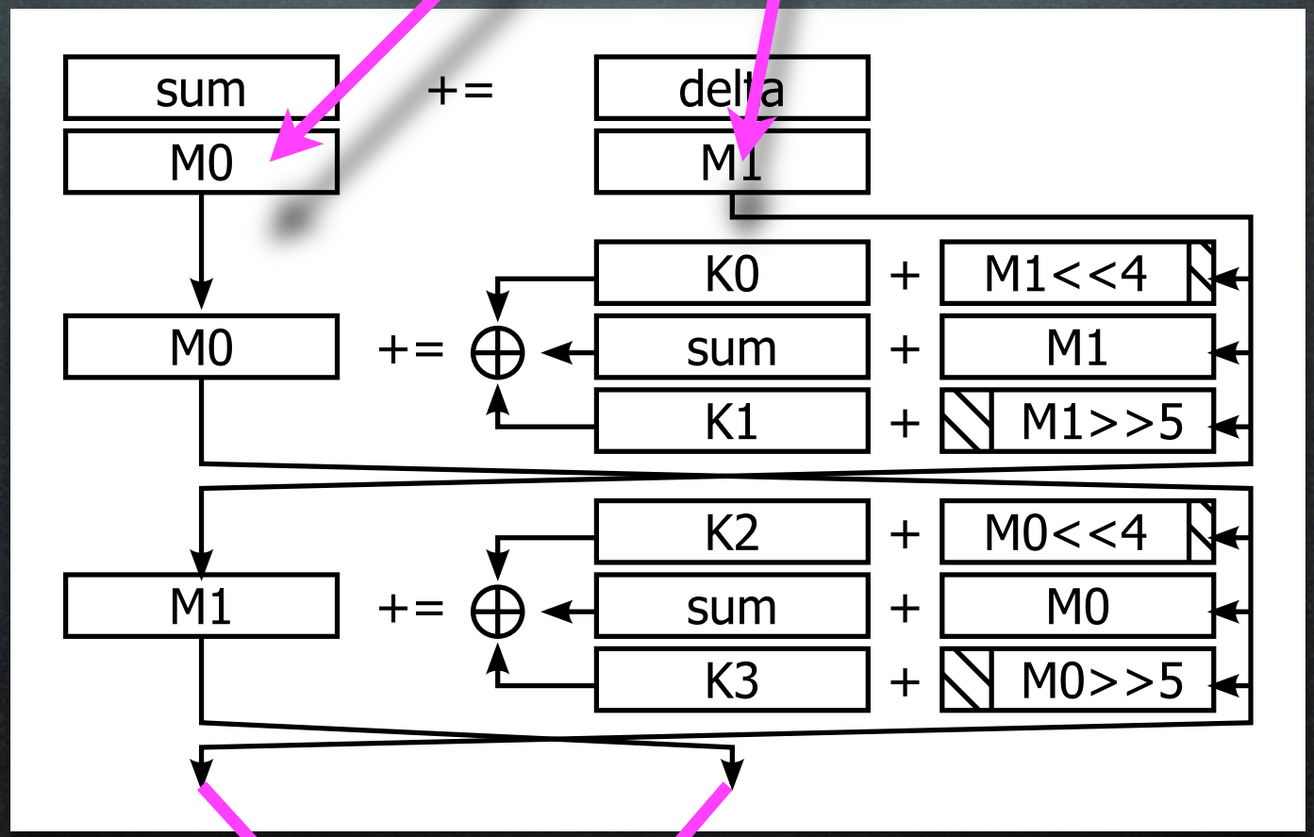


round one

(first block)

“hi h” | “ow a” | “re y” | “ou a” | “ll t” | “oday”

$c_1[0]$ $c_1[1]$



round two

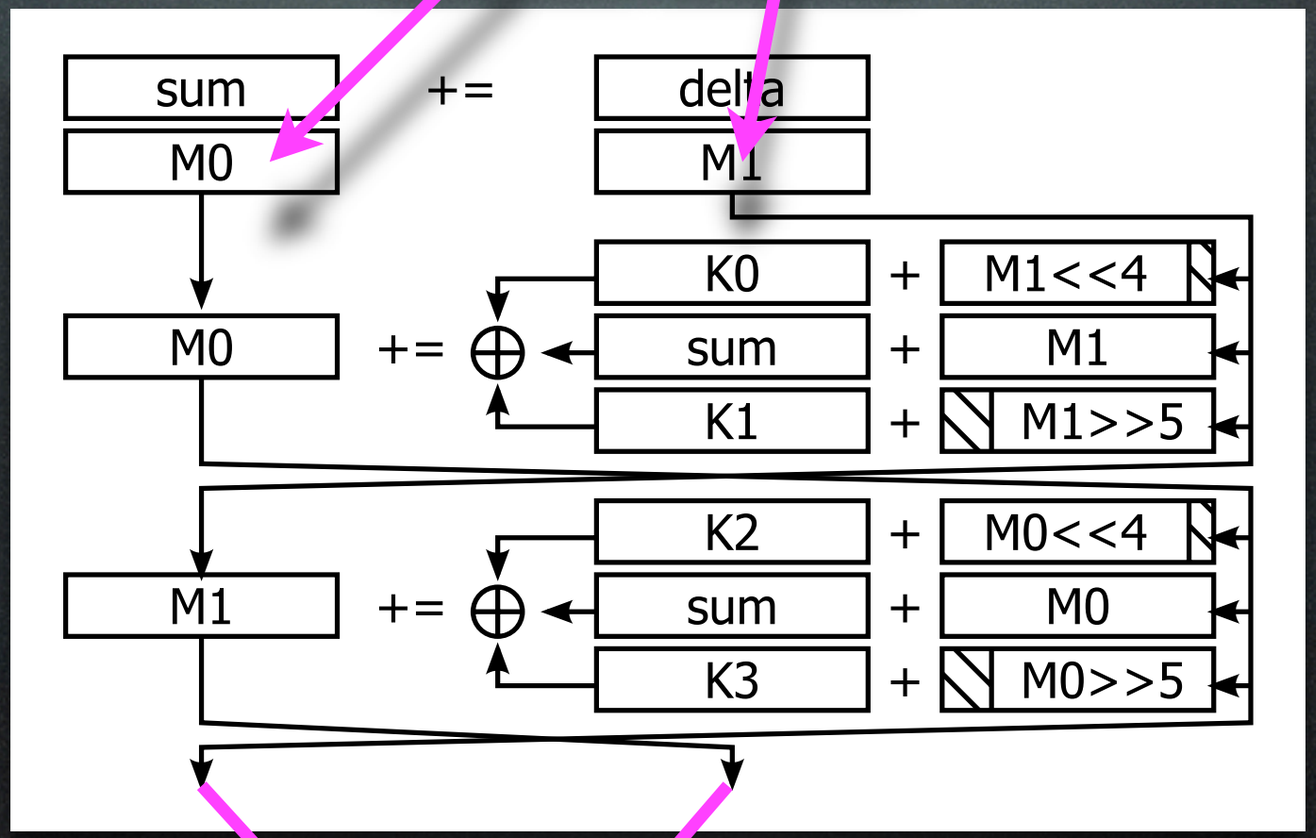
$c_2[0]$ $c_2[1]$

...

“hi h” | “ow a” | “re y” | “ou a” | “ll t” | “oday”

$c_{31}[0]$ $c_{31}[1]$

round 32



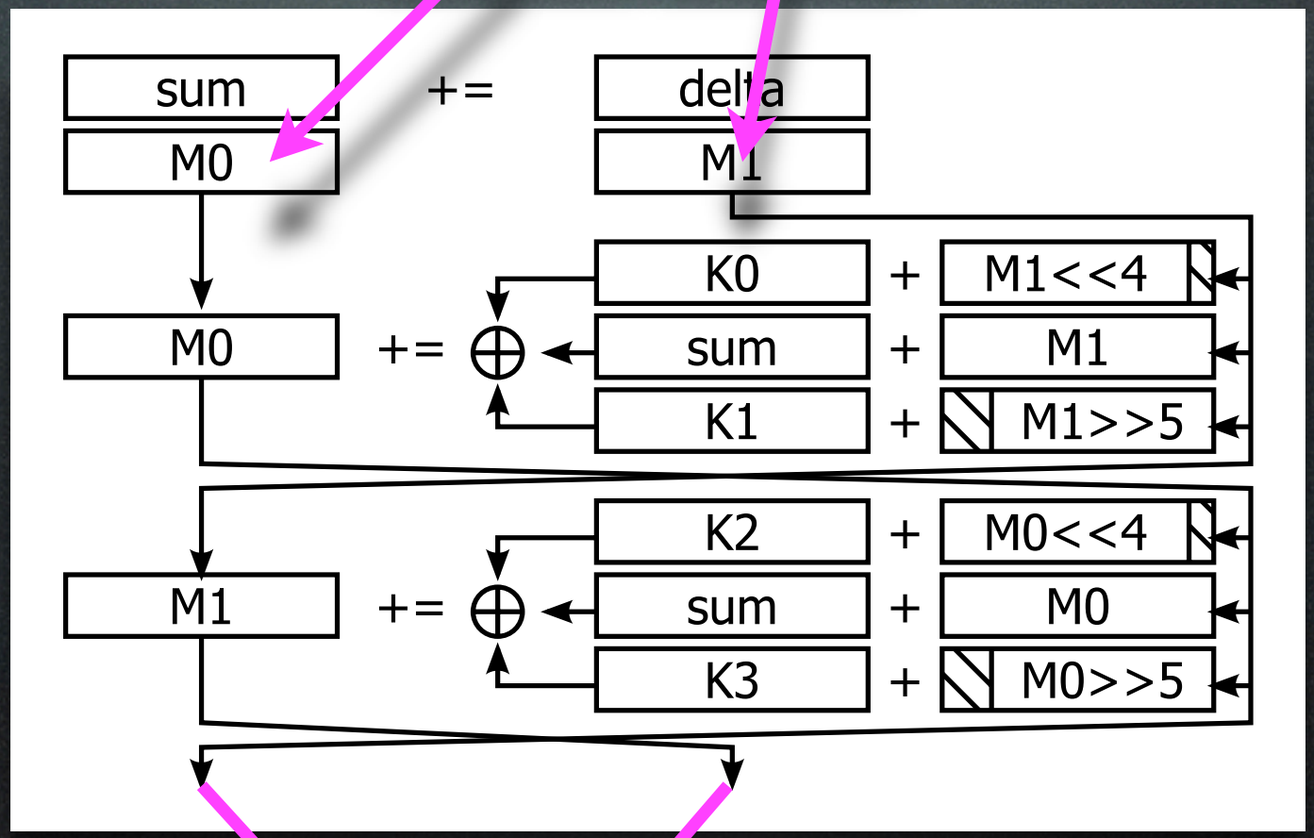
$c_{32}[0]$ $c_{32}[1]$

“♒♓ ♒” “□◆ ♋”

…

“hi h” | “ow a” | “re y” | “ou a” | “ll t” | “oday”

m[0] | m[1]



round one

$c_1[0]$ | $c_1[1]$

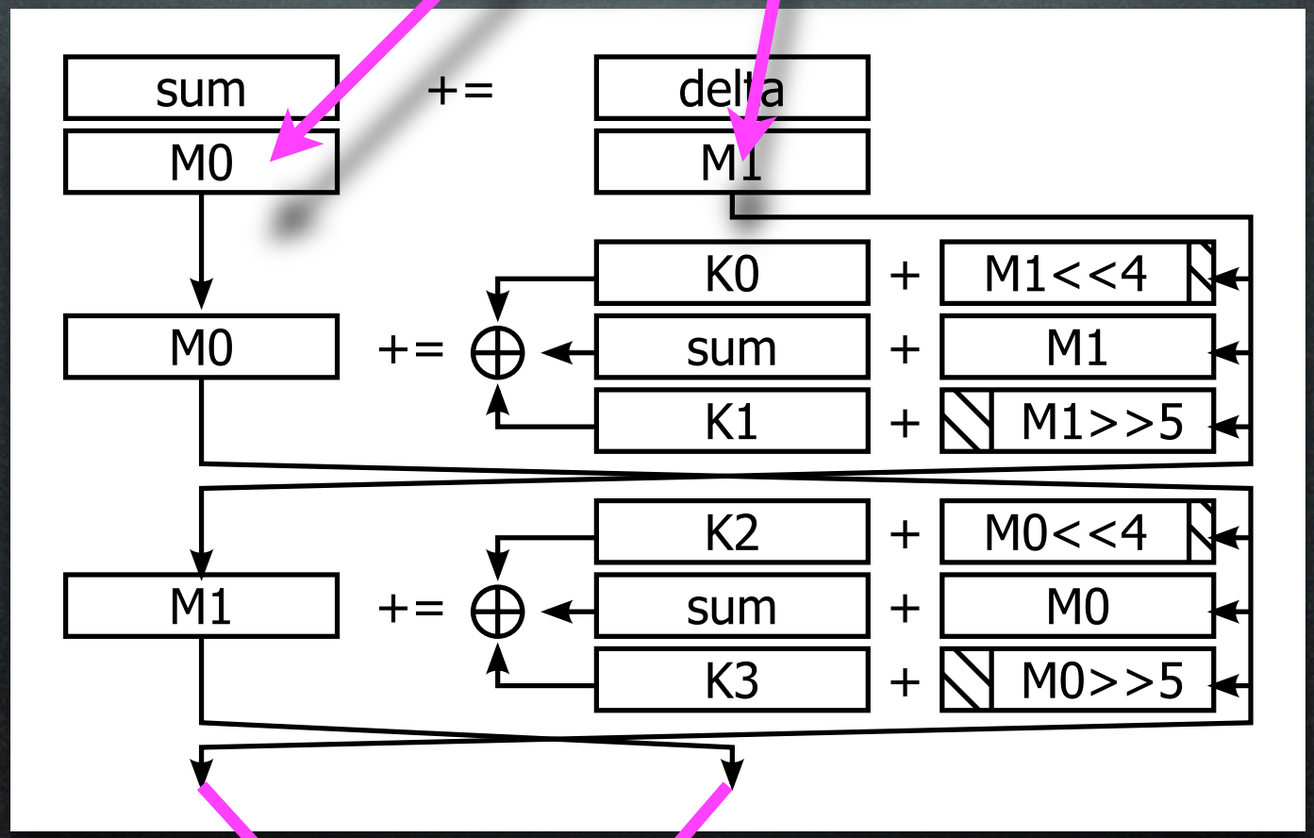
(next block)

“♒♓ ♒” | “□◆ ♋”

“hi h” | “ow a” | “re y” | “ou a” | “ll t” | “oday”

$c_{31}[0]$ $c_{31}[1]$

round 32



$c_{32}[0]$ $c_{32}[1]$

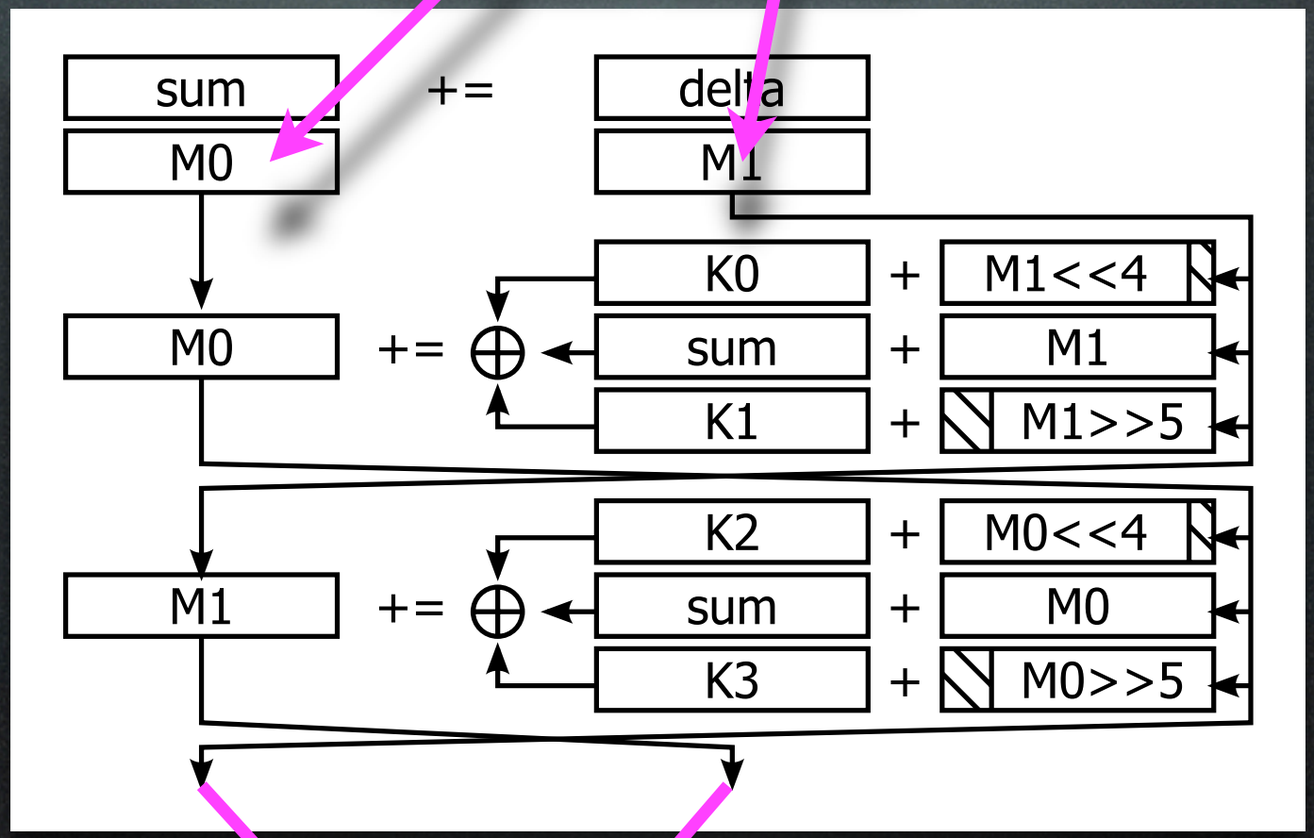
“♒♓ ♒” | “□◆ ♋” | “❒♏ ♲” | “□◆ ♋”

…

“hi h” | “ow a” | “re y” | “ou a” | “ll t” | “oday”

$c_{31}[0]$ $c_{31}[1]$

round 32



$c_{32}[0]$ $c_{32}[1]$

Etc.!

“♒♓ ♒” | “□♦ ♋” | “❒♏ ♲” | “□◆ ♋” | “●● ♦” | “□♎♋♲”

TEA

Tiny Encryption Algorithm

- We use one-round TEA for this project.

Agenda

- Problem Overview
  - Tiny Encryption Algorithm
  - Evolutionary Computation ←
- Experimental Design
- Results
- Conclusions

EC
Evolutionary Computation

- a whole family of *optimization techniques*
- all somewhat *inspired by biological evolution*
- we combine two techniques in this project
- *genetic algorithm* and *harmony search*

GA

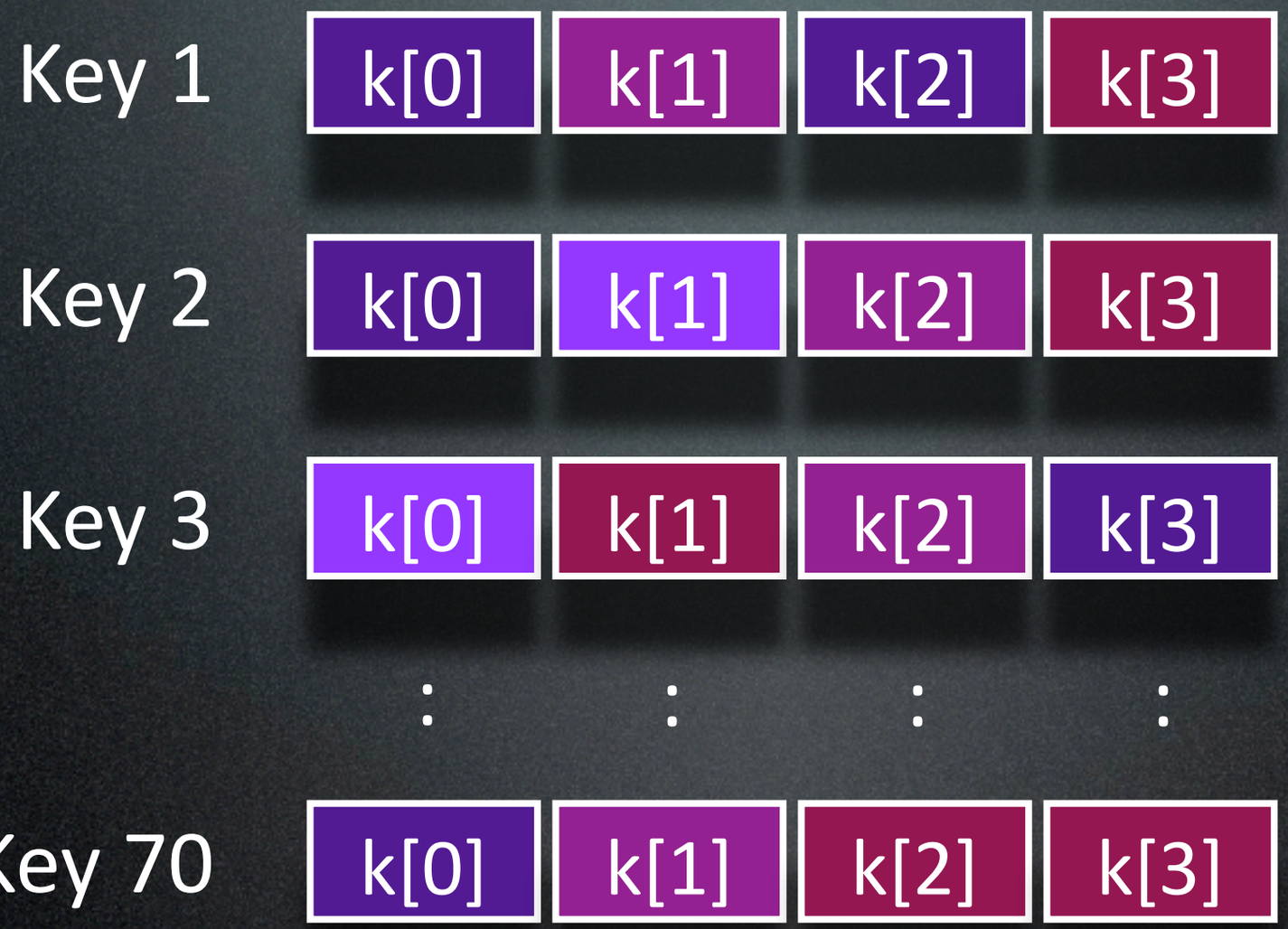
Genetic Algorithm

- search space represented by chromosomes
- optimize a population of chromosomes
- called *evolution*
- Let's discuss this in context of this project.

(Fraser, 1957)

GA

Genetic Algorithm • A Population



A population of seventy chromosomes (keys).

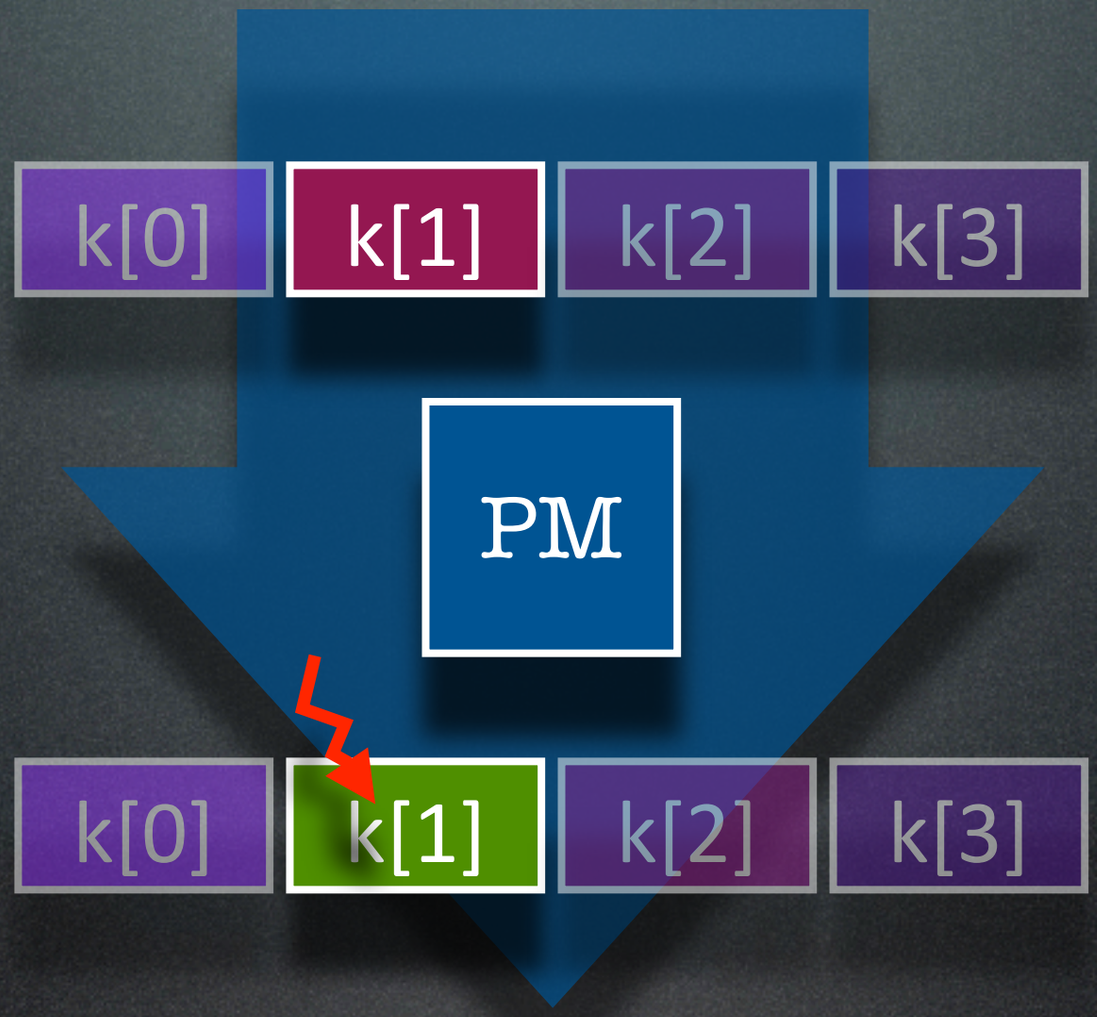
GA

Genetic Algorithm

- operations performed on population
- get better and better keys at each generation

GA

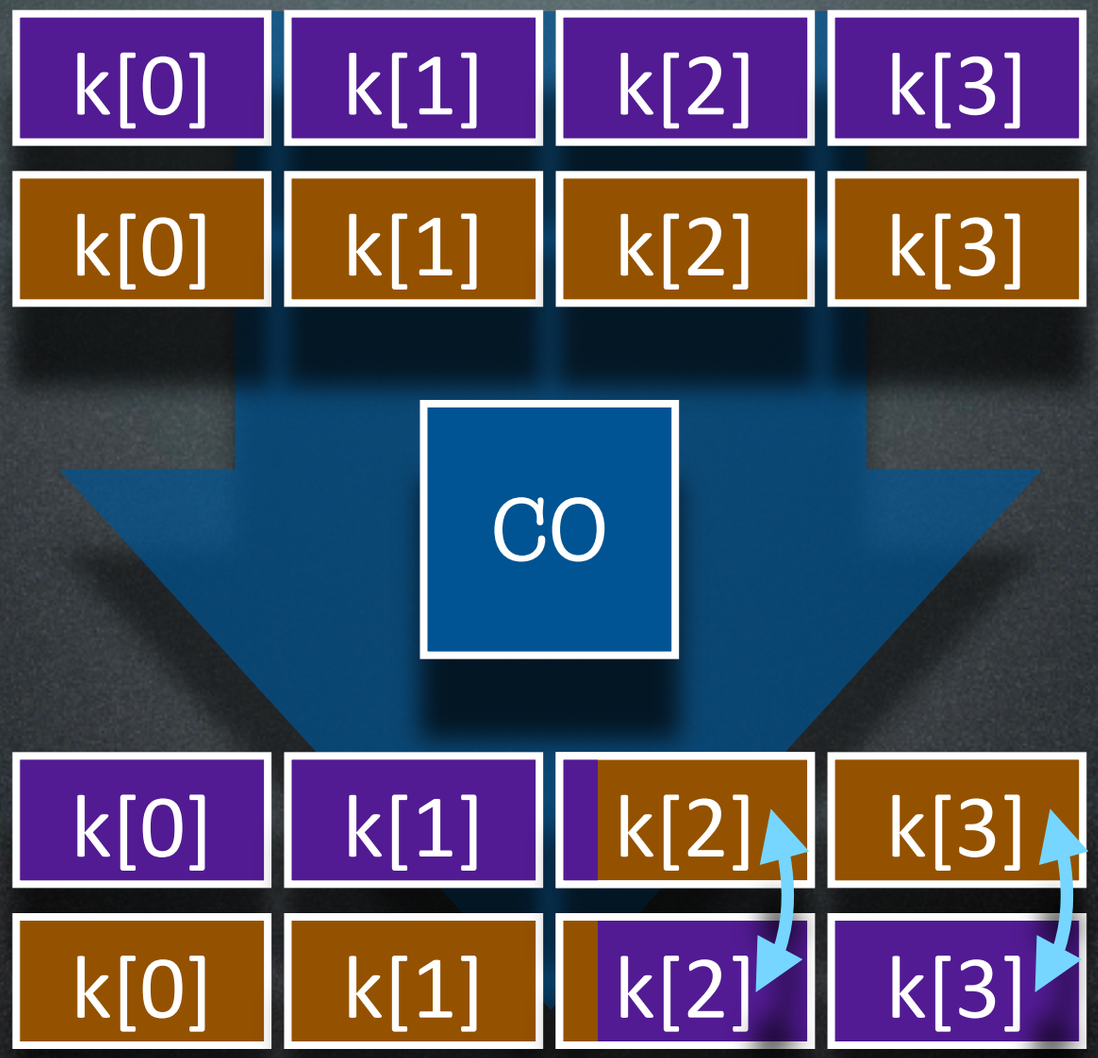
Genetic Algorithm • Point Mutation



Operation 1: We define a point mutation as a bit-flip.
(2% uniform pseudorandom probability for each bit)

GA

Genetic Algorithm • Crossover



Operation 2: Two parents yield two children in crossover.
(pairs of parents are randomly selected)

GA

Genetic Algorithm • Fitness Function

- select the fittest keys to continue
- unfit keys removed from population
- want: fitness should improve over time
- our fitness function is a hamming distance

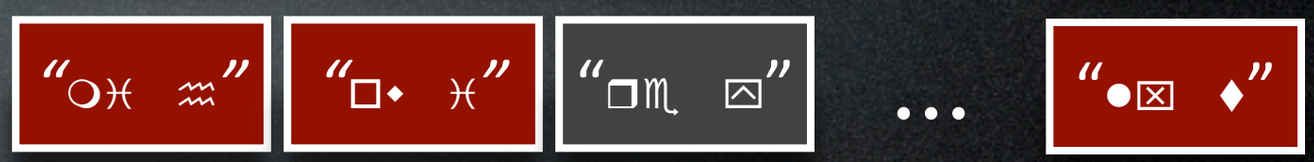
GA

Genetic Algorithm • Fitness Function

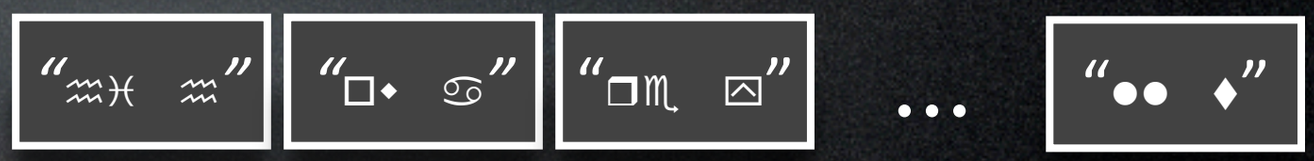
Key 1's ciphertext

Key 2's ciphertext

Key 3's ciphertext



Correct Key's ciphertext



Hamming distance calculated for each key's ciphertext against the correct key's ciphertext.

HS

Harmony Search

- Let’s discuss this as a modification to GA.
- two more operators

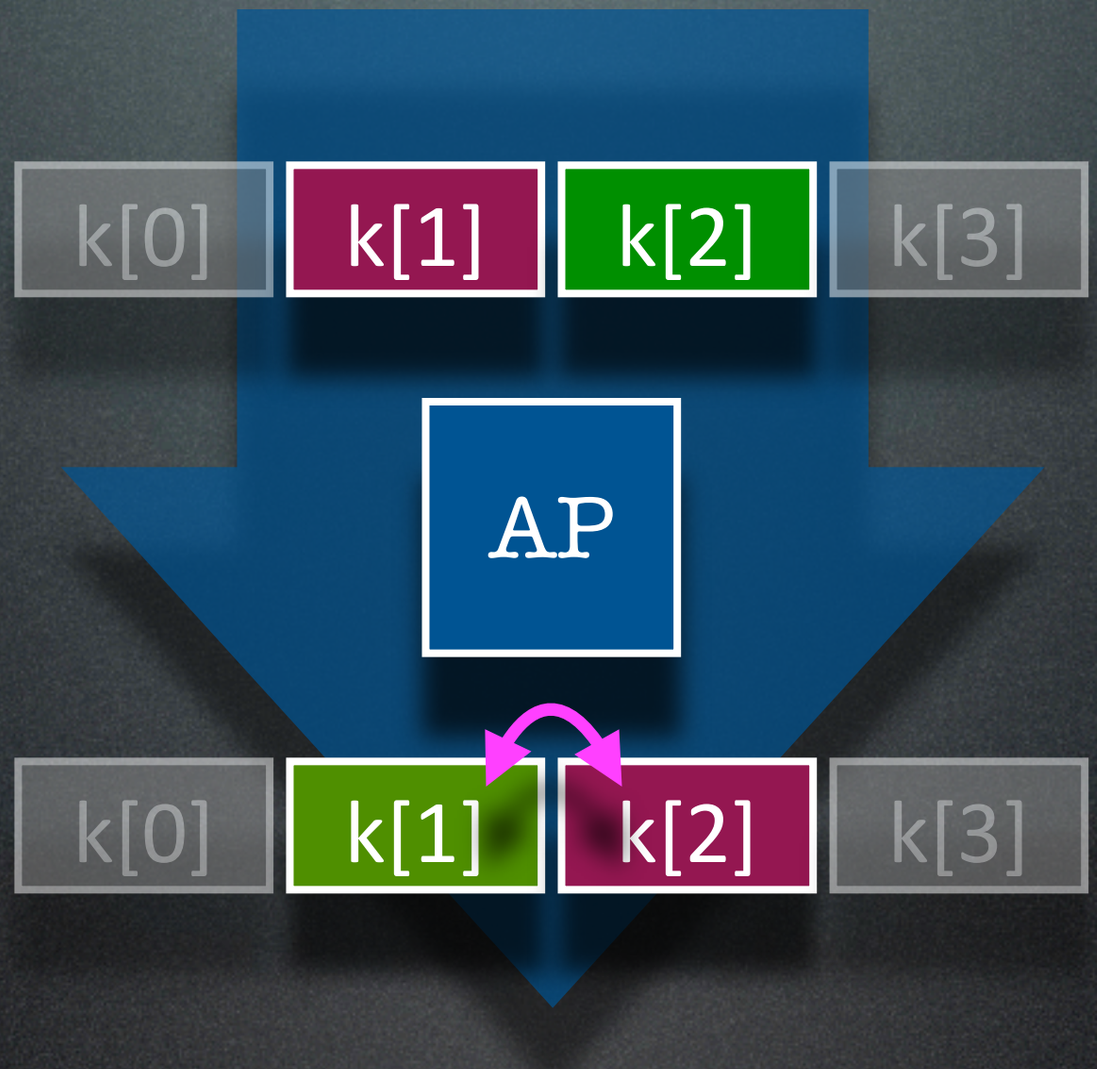
Harmony Search is inspired by *musical improvisation*
(Geem & Kim, 2001)

HS Harmony Search • Improvise Harmony

Operation 3: Entire population used to improvise harmony. (contributing parents randomly selected)

HS

Harmony Search • Adjust Pitch



Operation 4: We define adjust pitch as byte or word swaps. (pairs of bytes or words randomly selected within a key)

Agenda

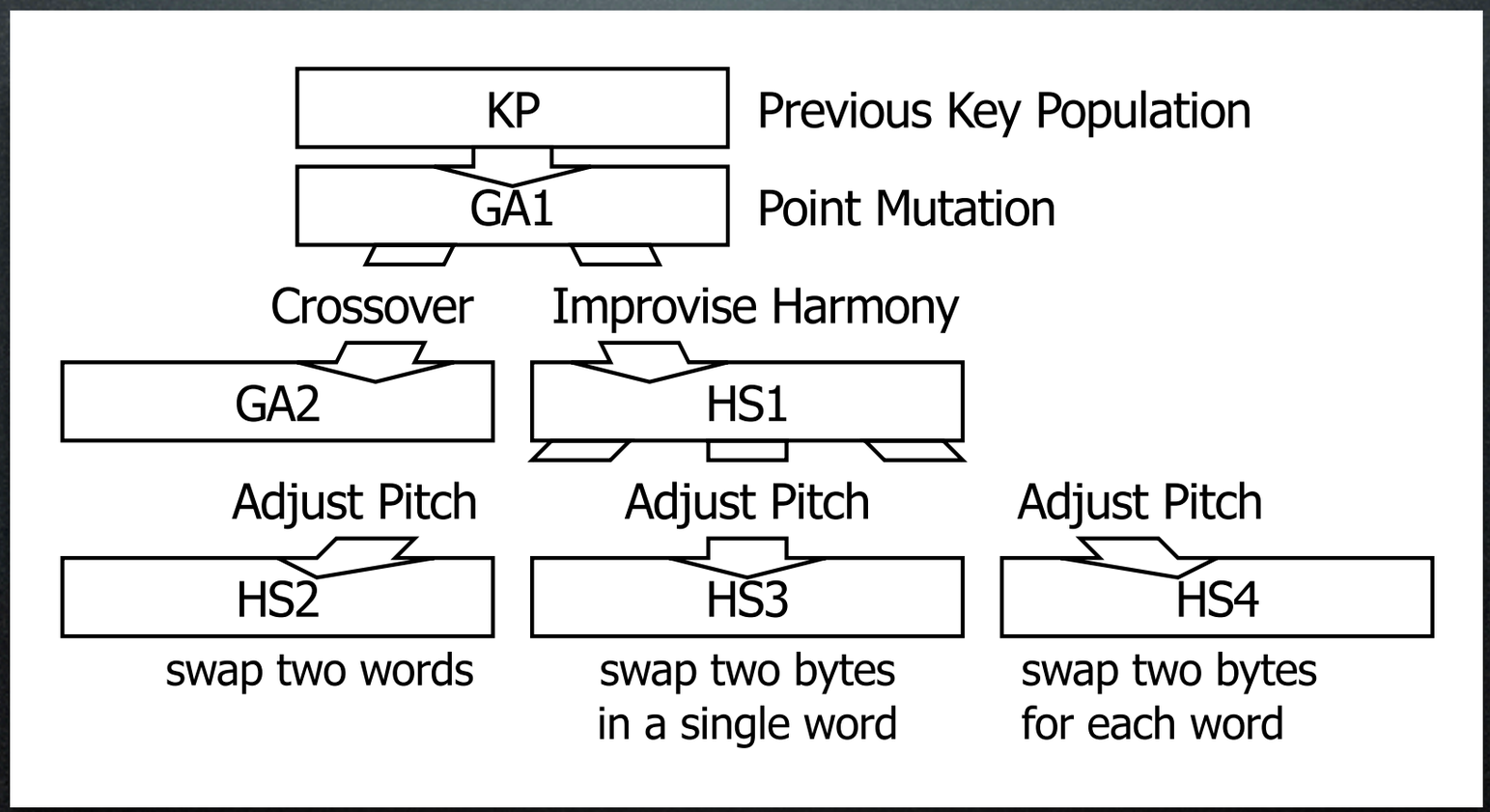
- Problem Overview
  - Tiny Encryption Algorithm
  - Evolutionary Computation
- Experimental Design ←
- Results
- Conclusions

Experimental Design

- We will now explain ...
  - the arrangement of *operators* in the EC
  - the *selection* of chromosomes in the EC
  - selection of known plaintexts and keys

EC Operators & Selection

Operating a population of keys in a generation ...



Each box is a reservoir of 70 keys.

After all operations are complete, the best ten from each box are advanced to next generation.

- Let’s talk about how we selected plaintexts

Plaintexts and Keys

- Plaintexts were chosen at random with a pseudorandom uniform number generator.
- Ciphertexts were calculated using one-round TEA.
- 100 plaintext message blocks per trial
- 30 trials per experiment

Plaintexts and Keys

- two experiments run given two keyschemes
  - scheme 1: create keys with the words ...

    {0x00000000, 0xFFFFFFFF, 0xXXXXXXXX}†

  - scheme 2: create keys with the words ...

    {0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF}

†0xXXXXXXXX is four random bytes.

Plaintexts and Keys

- Number of keys in keyscheme 1:
  - 3^4 = 81 schemes
- Number of keys in keyscheme 2:
  - 4^4 = 256 schemes
- *Every single scheme* was committed to 30 trials of randomly generated plaintexts.

Agenda

- Problem Overview
  - Tiny Encryption Algorithm
  - Evolutionary Computation
- Experimental Design
- Results ←
- Conclusions

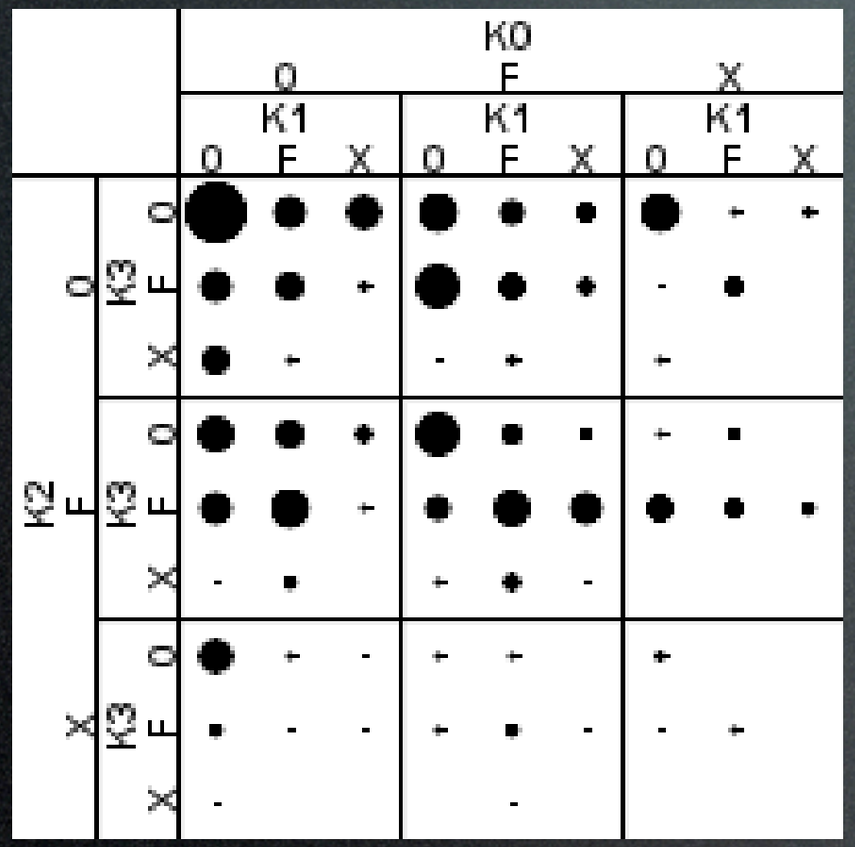
Results

Experiment 1

{0x00000000, 0xFFFFFFFF, 0xXXXXXXXX}

Results

Experiment 1 - Summary Plots



Proportion of Convergences

- really easy:
  - >2 zeros
- easy:
  - (K0, K1) = 0 or F
  - (K2, K3) = 0 or F
- hard:
  - >1 X (random)

Results

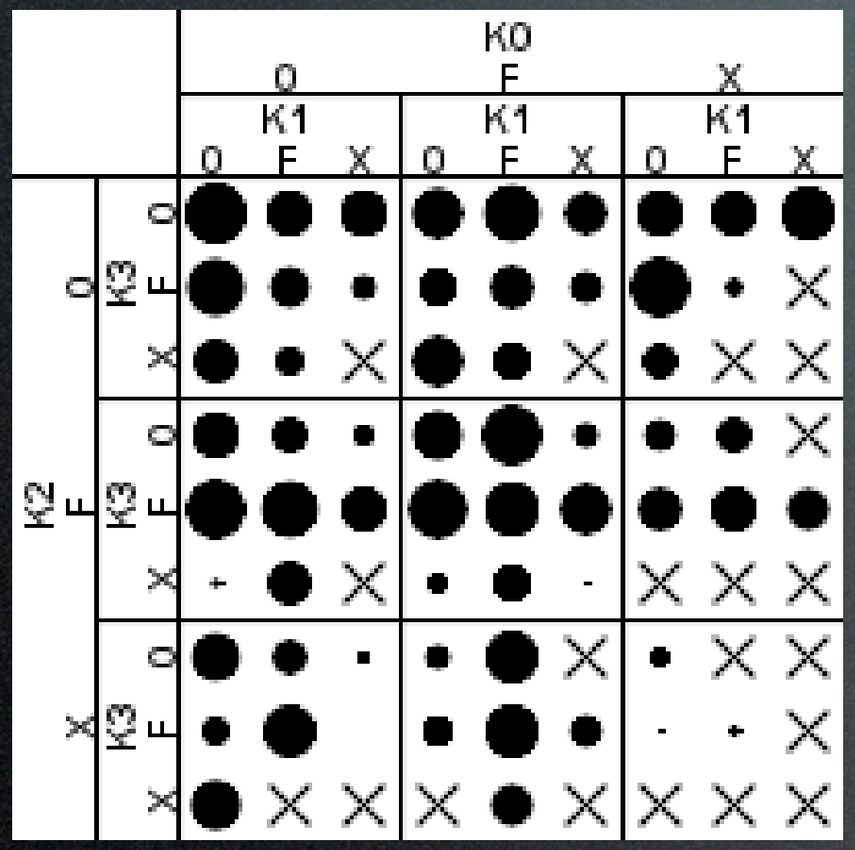
Experiment 1 - Number of Convergences

Word	Occurrences	Affected keys	$\bar{x}$ / 30	σ
0	1	32	2.9	3.9
	2	24	5.0	5.1
	3	8	13.1	1.5
	4	1	24.0	–
F	1	32	2.8	4.4
	2	24	5.6	5.0
	3	8	9.0	3.3
	4	1	14.0	–
X	1	32	5.1	4.0
	2	24	0.9	1.3
	3	8	0.0	0.0
	4	1	0.0	–

Average number of convergences over 30 trials
(with standard deviation)

Results

Experiment 1 - Summary Plots



Speed of Convergence

- larger dot is earlier
- cross = no convergence
- same pattern as before

Results

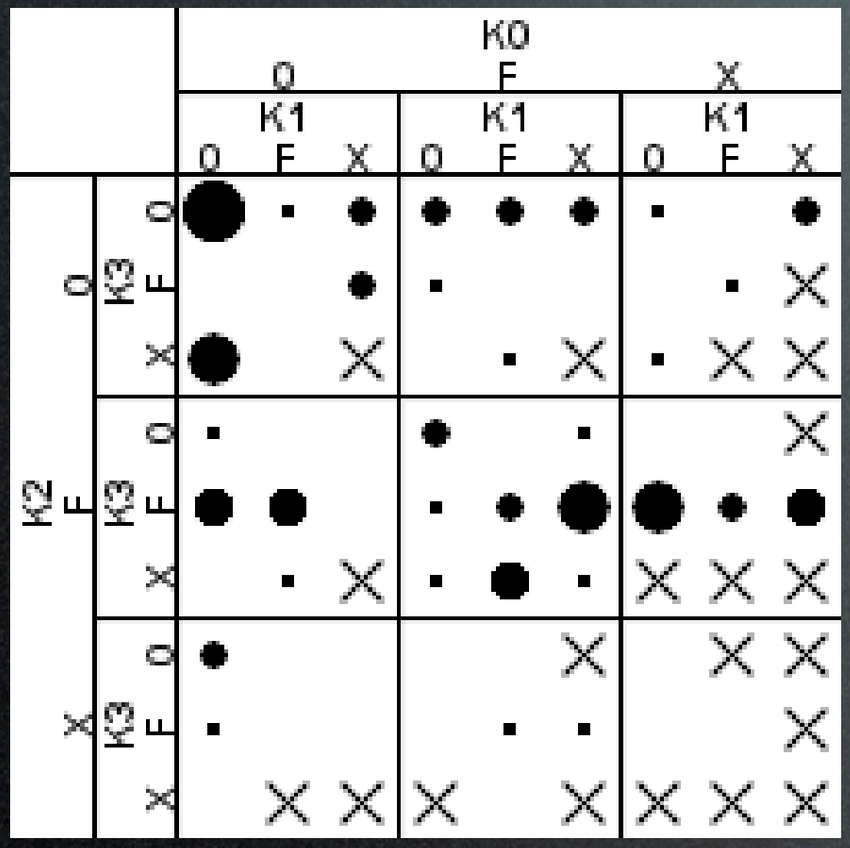
Experiment 1 - Convergence Generation

Word	Occurrences	Converged trials	$\bar{x}$ generation	σ
0	1	92 / 960	2069.6	1519.1
	2	121 / 720	2192.8	1399.7
	3	105 / 240	1699.0	1311.8
	4	24 / 30	801.0	1112.8
F	1	89 / 960	2118.6	1504.9
	2	135 / 720	2220.5	1370.9
	3	72 / 240	1506.2	1334.5
	4	14 / 30	1216.7	1357.5
X	1	164 / 960	2241.4	1435.5
	2	22 / 720	2931.8	1466.5

Average generation of convergence over 30 trials
(with standard deviation)

Results

Experiment 1 - Summary Plots



Proportion of Degenerate Keys

- TEA degenerate keys
- each key is part of an equivalent triplet
- (K0, K1) = 0 or F
- (K2, K3) = 0 or F
- large number of random breaks: {(000X), (FXFF), (FFFX), (XXFF)}

Results

Experiment 2

{0x<u>FF</u>000000, 0x00<u>FF</u>0000, 0x0000<u>FF</u>00, 0x000000<u>FF</u>}

Results

Experiment 2 - Number of Convergences

Arrangement of matching words	Affected keys	$\bar{x}$ / 30	σ
$K0 = K1 \wedge K2 = K3$	16	7.0	4.4
$K0 = K1 \wedge K2 \neq K3$	48	6.9	3.7
$K0 \neq K1 \wedge K2 = K3$	48	6.9	4.1
$K0 \neq K1 \wedge K2 \neq K3$	144	7.0	3.2
$K0 = K2 \wedge K1 = K3$	16	10.8	2.8
$K0 = K2 \wedge K1 \neq K3$	48	8.3	2.8
$K0 \neq K2 \wedge K1 = K3$	48	8.3	3.6
$K0 \neq K2 \wedge K1 \neq K3$	144	5.7	3.2
exactly four matching words	4	13.0	3.4
exactly three matching words	48	9.8	3.2
exactly two matching words	180	6.5	3.1
no matching words	24	3.5	1.7

Average number of convergences over 30 trials
(with standard deviation)

Results

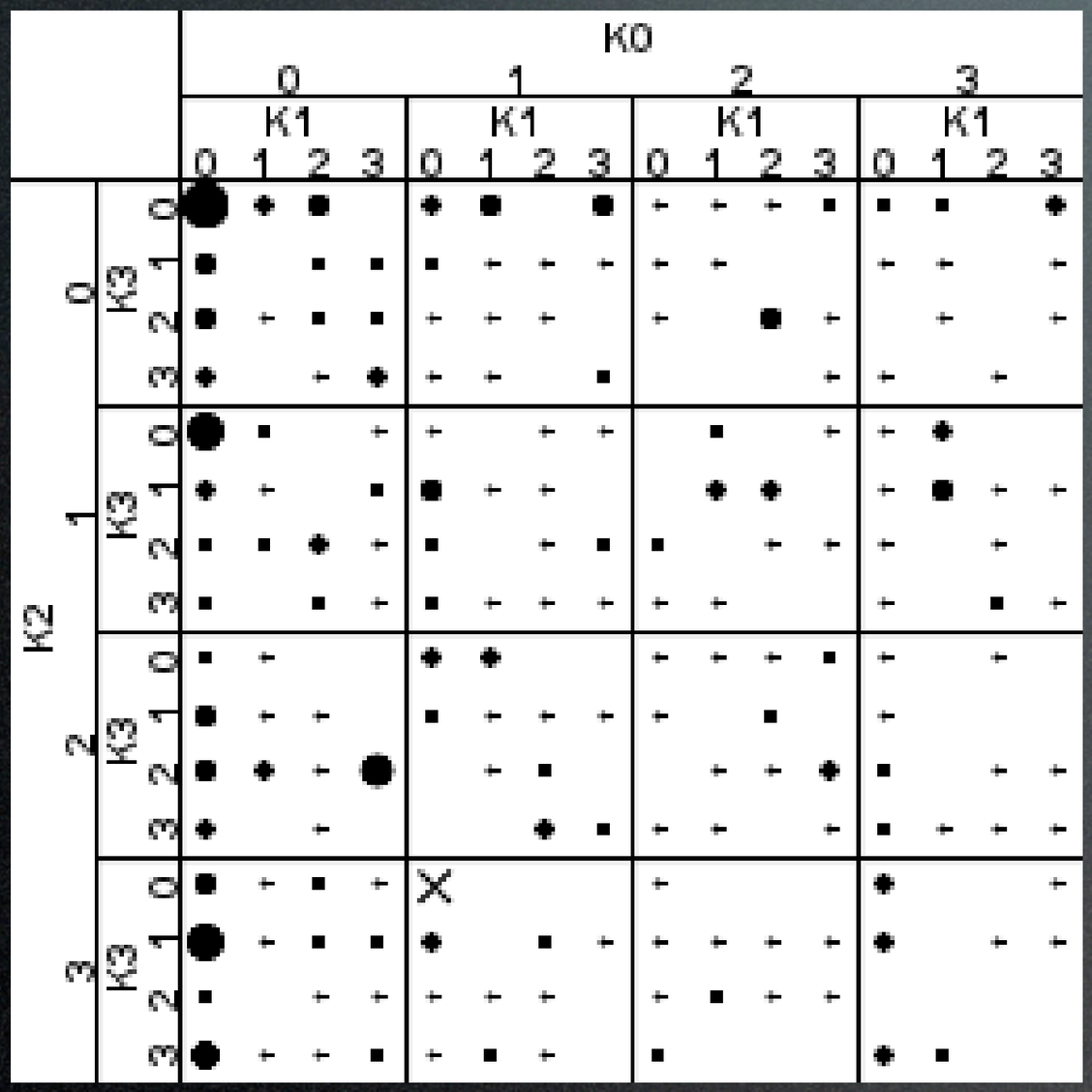
Experiment 2 - Convergence Generation

Word	Converged trials	$\bar{x}$ generation	σ
exactly four matching words	52 / 120	1779.7	1352.0
exactly three matching words	471 / 1440	2196.9	1348.2
exactly two matching words	1175 / 5400	2408.9	1291.0
no matching words	85 / 720	2776.8	1207.8

Average generation of convergence over 30 trials
(with standard deviation)

Results

Experiment 2 - Summary Plots



Proportion of Degenerate Keys

- large degenerate areas
  - (K0, K1) = FF00000
  - (K2, K3) = FF00000

Agenda

- Problem Overview
  - Tiny Encryption Algorithm
  - Evolutionary Computation
- Experimental Design
- Results
- Conclusions ←

Conclusions

Experiment 1

{0x00000000, 0xFFFFFFFF, 0xXXXXXXXX}

- more random words implies more resilience
- all-on-bit words more resilient to all-off-bit
- EC method can find equivalent keys

Conclusions

Experiment 2

{0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF}

- matched words in (K0, K2) and (K1, K3) easier
- more matched words even easier (due to HS)
- equivalent keys easiest to derive for words with {0xFF000000} in (K0, K1) or (K2, K3).

Conclusions

- EC methods capable of attacking one-round Feistel block ciphers.
- HS is particularly good at probing solutions with repetitions.
- Equivalent keys can also be derived.

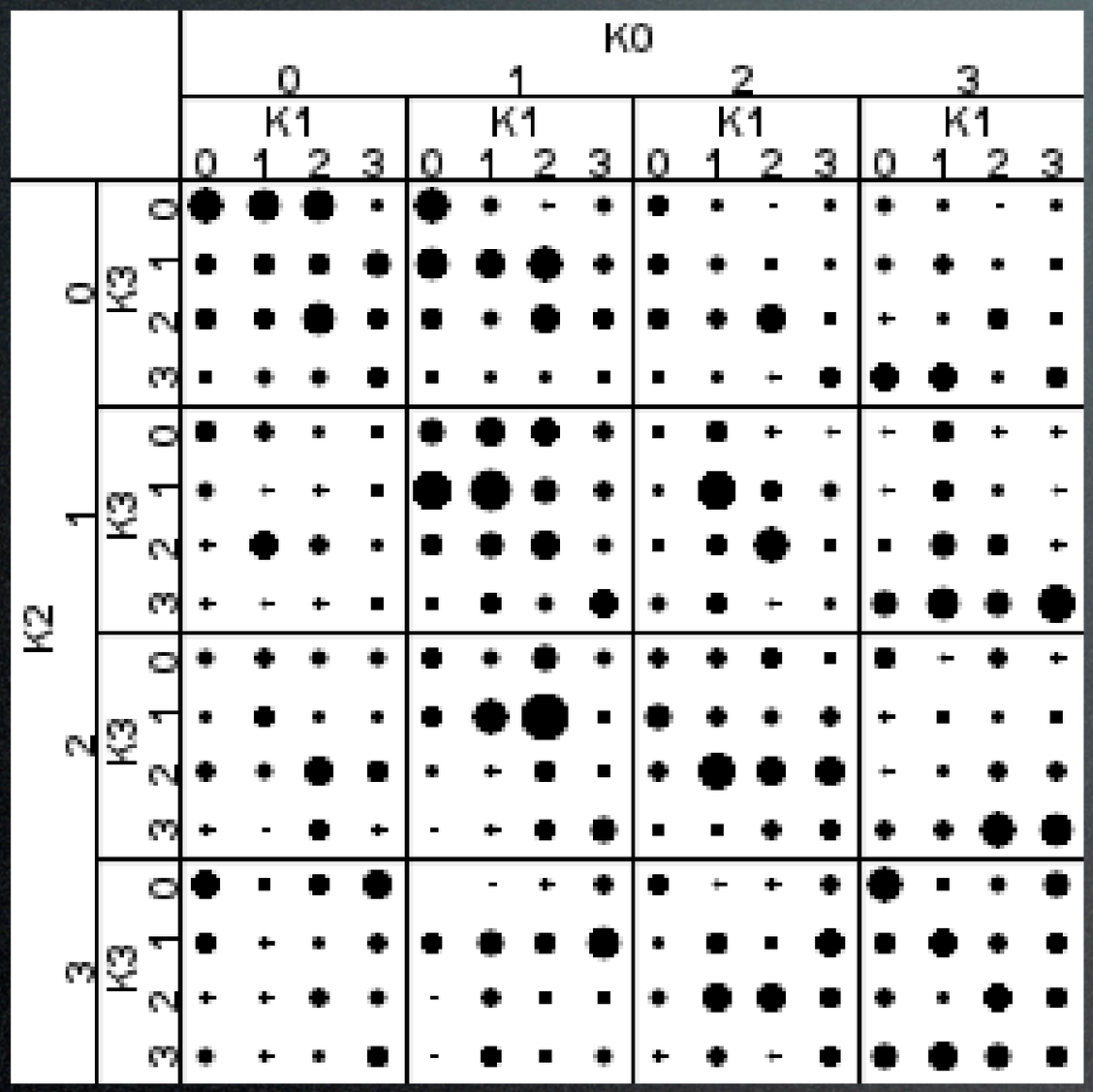
Thanks for listening!

http://eddiema.ca

ema@uoguelph.ca

Results

Experiment 2 - Summary Plots

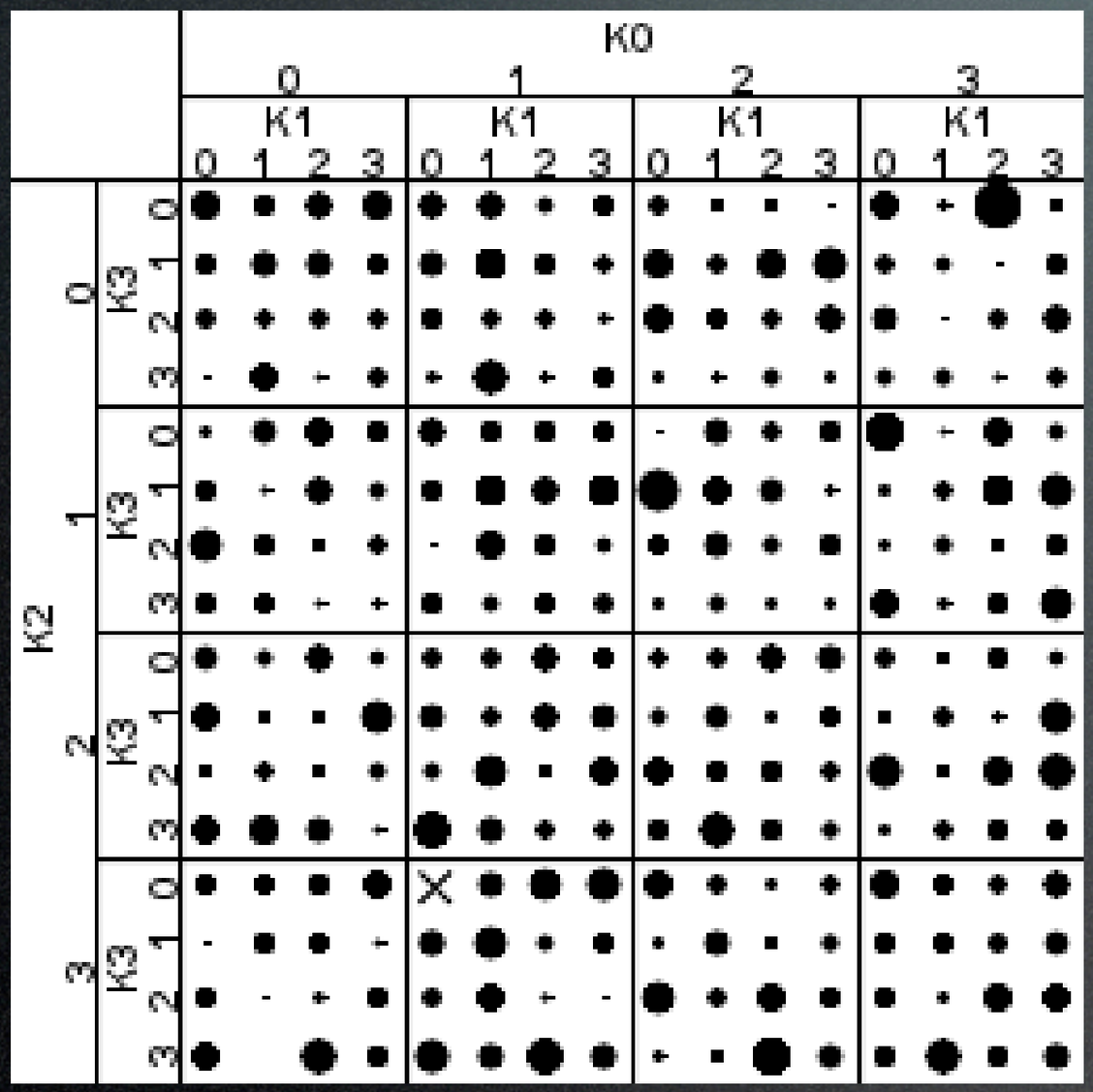


Proportion of Convergences

- no obvious pattern

Results

Experiment 2 - Summary Plots



Speed of Convergence

- no obvious pattern