# A Large Term Rewrite System Modelling a Pioneering Cryptographic Algorithm

**Hubert Garavel    Lina Marsso**

**Inria Grenoble – LIG**

**Université Grenoble Alpes**

http://convecs.inria.fr

# Outline

- 1. Introduction to Term Rewrite Systems (TRS)
- 2. The Message Authenticator Algorithm (MAA)
- 3. Earlier Models of the MAA
- 4. Formal Modelling of the MAA as a TRS
- 5. Validation of the MAA Model
- 6. Conclusion

# 1. Introduction to Term Rewrite Systems (TRS)

# Term Rewrite Systems (TRS)

- A fundamental means to express computation
- Basic concepts:
  - **sorts**: abstract data domains
  - **operations**: take *N* arguments and return one result
  - **terms**: algebraic expressions (operations, free variables)
  - **rewrite rules**: *left-hand term* $\to$ *right-hand term*

    not (and (A, B)) $\to$ or (not (A), not (B))
- Used in specification/programming languages
  - **algebraic**: abstract data types
  - **functional**: constructor types and pattern matching

# Where can one find TRS models?

- Paradox:
  - abundant literature on the theory of TRS
  - but difficult to find TRS models of realistic problems
- Available TRS models:
  - Rewrite Engines Contests (2006, 2008, 2010)
    the largest models have at most 300 lines
  - Specification of languages / compilers using TRS
    models can be large (10,000+ lines) but they are not "pure" TRS (they use strategies, sub-sorts, etc.)
- This talk: a large TRS modelling a cryptographic algorithm

# The REC Language

- REC: a textual notation for TRS models
- Introduced during the 2nd REC contest (2008)
  - human-readable, tool-independent format
  - supports strong typing (many-sorted specifications)
  - supports conditional rewrite rules (Boolean guards)
- We use a slightly enhanced version of REC
  - added distinction: constructors vs non-constructors
  - a few restrictions: left-linear rules, no equations between constructors, etc.
  - automatically translated into 13 different languages

# Example 1: Booleans in REC

```
SORTS
  Bool                                  <- - - - - - - - - - - - - - -  abstract data domain
CONS
  false : -> Bool                       <- - - - - - - - - -  primitive operations
  true : -> Bool                                              (constructors)
OPNS
  andBool : Bool Bool -> Bool           <- - - -  defined operations
  orBool : Bool Bool -> Bool                      (non-constructors)
VARS
  L : Bool                              <- - - - - - - - - - - - - - -  free variables
RULES
  andBool (false, L) -> false           <- - - -  rewrite rules that
  andBool (true, L) -> L                          define non-constructors

  orBool (false, L) -> L
  orBool (true, L) -> true
```

# Example 2: Naturals in REC (1/2)

```
SORTS
  Nat
CONS
  zero : -> Nat
  succ : Nat -> Nat
OPNS
  addNat : Nat Nat -> Nat
  multNat : Nat Nat -> Nat
  eqNat : Nat Nat -> Bool
  ltNat : Nat Nat -> Bool
VARS
  N N' : Nat
```

# Example 2: Naturals in REC (2/2)

```
RULES
  addNat (N, zero) -> N
  addNat (N, succ (N')) -> addNat (succ (N), N')

  multNat (N, zero) -> zero
  multNat (N, succ (N')) -> addNat (N, multNat (N, N'))

  eqNat (zero, zero) -> true
  eqNat (zero, succ (N')) -> false
  eqNat (succ (N), zero) -> false
  eqNat (succ (N), succ (N')) -> eqNat (N, N')

  ltNat (zero, zero) -> false
  ltNat (zero, succ (N')) -> true
  ltNat (succ (N'), zero) -> false
  ltNat (succ (N), succ (N')) -> ltNat (N, N')
```

# 2. The Message Authenticator Algorithm (MAA)

# Cryptography basics

- Message Digest
  - function: (long) message $\rightarrow$ (short) numeric value
  - ensures integrity (the message has not been modified)
  - example: MD5
- Message Authentication Code (MAC)
  - function: (long) message, (short) key $\rightarrow$ (short) value
  - the key is secret, shared by the sender and the receiver
  - ensures both authentication and integrity
  - examples: hash-based (HMAC) , universal (UMAC), block ciphers (CMAC, OMAC, PMAC), etc.

# Message Authenticator Algorithm (MAA)

- First widely-used MAC function
- Designed by Donald Davies and David Clayden (NPL, 1983)
  - to protect banking transactions
  - intended to be implemented in software (32-bit PCs)
- Adopted by financial institutions
  - standardized by ISO in 1987 [ISO 8730 and 8731-2]
  - attacks published in the mid 90s
  - withdrawn from ISO standards in 2002

# Overview of the MAA

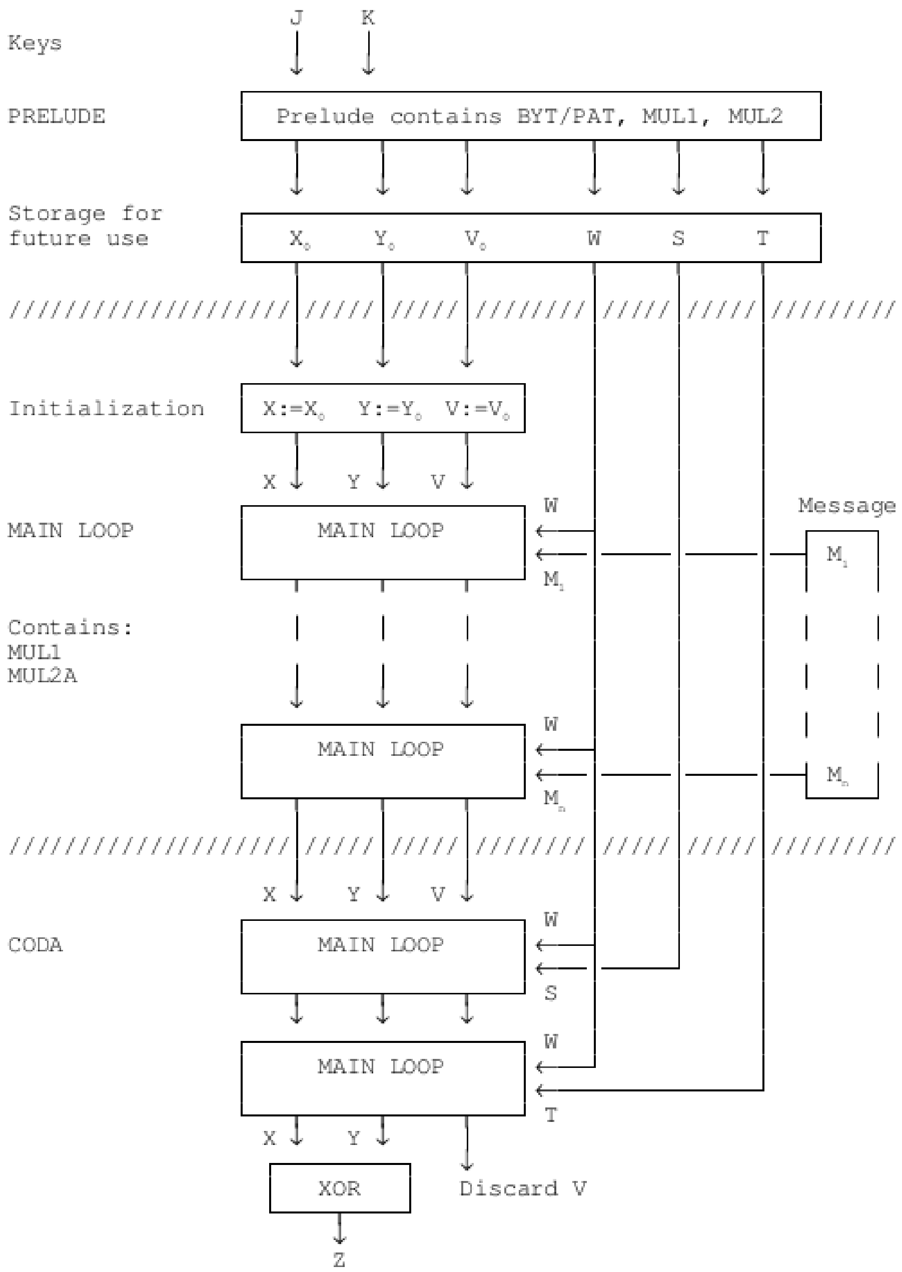
- Inputs:
  - A 64-bit key (split into two blocks J, K)
  - A message, seen as a sequence of blocks (message should be less than 1,000,000 blocks)
- Outputs:
  - A 32-bit MAC value (much too short nowadays!)
- Basic operations:
  - logical: AND, OR, XOR, CYC (bit rotation)
  - arithmetic: ADD, MUL (mod $2^{32}$), MUL1 (mod $2^{32}$-1), MUL2 (mod $2^{32}$-2), MUL2A (faster variant of MUL2)

# MAA data flow

Prelude: converts key (J, K) into 6 blocks X0, Y0, V0, W, S, T

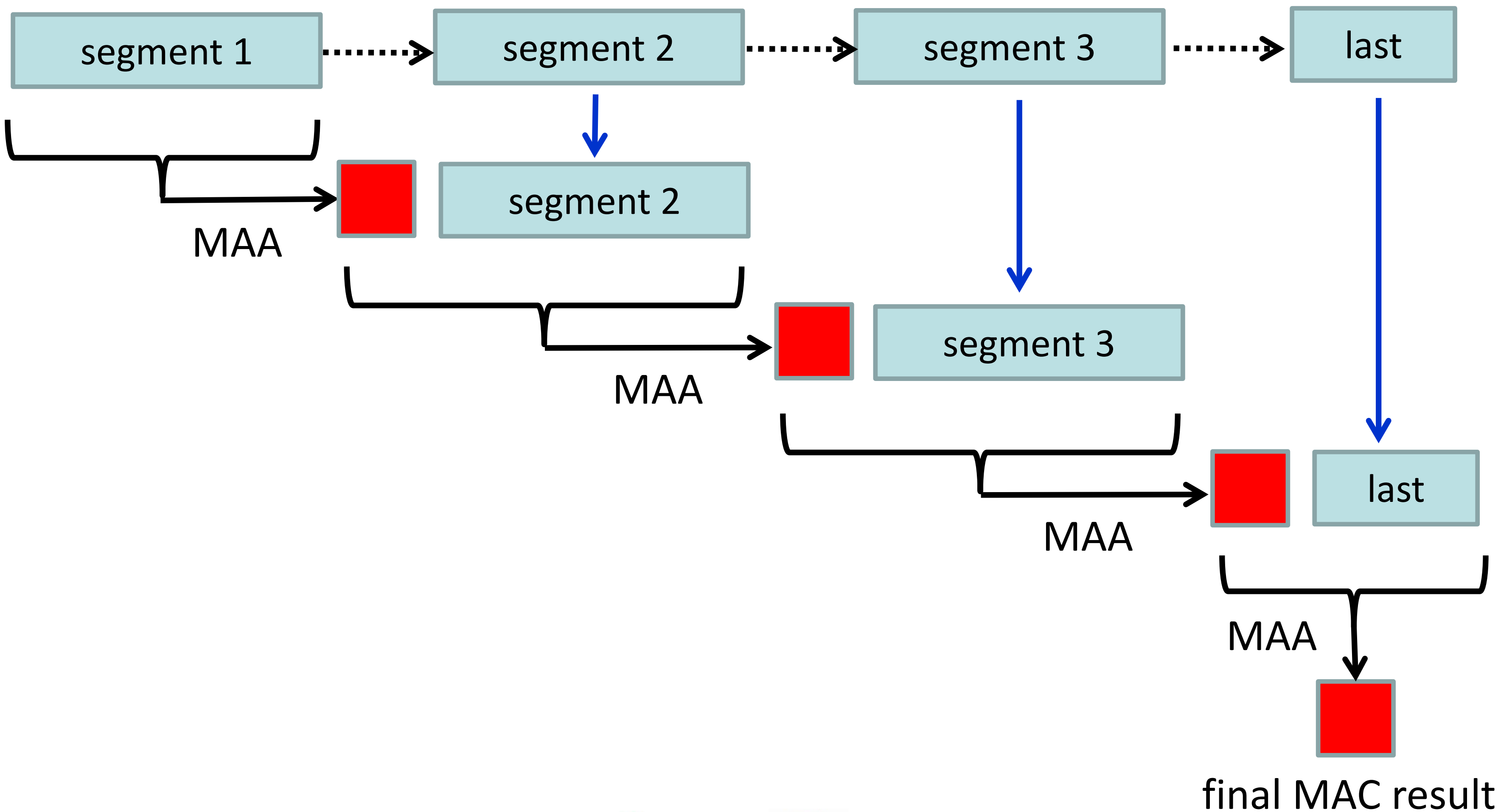
Main Loop: iterates on each message block, modifying 3 variables X, Y, V

Coda: two final iterations on the two blocks S and T



# "Mode of operation"

- Message is split into a list of 256-block segments



# 3. Earlier Models of the MAA

# Why choosing the MAA?

- More challenging than conventional examples:
  - protocols deal with simple data types
  - compilers deal with abstract syntax trees (explored using standard traversals)
  - cryptographic functions exhibit "strange" behavior by performing "irregular" calculations
- Large example, still of manageable complexity
- Definition of MAA is stable and available
- MAA played a role in the history of formal methods

# Informal specifications

- [Davies-Clayden-88] NPL technical report
  - complete definition of the MAA
  - gives two implementations in C and BASIC
  - these implementation do not support "mode of operation" (only work for messages <= 256 blocks)
- [ISO standard 8731-2]
  - core part very similar to [Davies-Clayden-88]
- These definitions in natural language are ambiguous at several places
  - e.g. byte ordering, mode of operation

# Formal specifications (1/2)

- NPL chose MAA to assess formal methods
  - they developed 3 formal specifications of the MAA

- 1) VDM [G. I. Parkin and G. O'Neill, 1990]
  - included as Annex B of ISO standard 8731-2:1992
  - 3 implementations derived manually from VDM: C, Miranda, Modula-2
- 2) Z [M. K. F. Lai, 1991]
  - Knuth's "literate programming" approach
  - Z code fragments inserted in natural-language ISO text

# Formal specifications (2/2)

- 3) LOTOS abstract data types [H. Munster, 1991]
  - fully formal, but non executable
  - "wishful thinking" equations: "given x, the result is y such that x = f (y)" ⇒ requires to invert function f
- 4) LOTOS abstract data types [H. Garavel, Ph. Turlier, 1992]
  - derived from [Munster-91]
  - rewritten to remove "wishful thinking" equations
  - a few types and functions implemented directly in C
  - implementation automatically derived (CAESAR.ADT)

# Goals of our work

Provide a model of the MAA in REC language with (at least) five qualities:

| | |
|---|---|
| ■ Formal | (no natural language) |
| ■ Exhaustive | (the full MAA is described) |
| ■ Self-contained | (no external C code) |
| ■ Validated | (correctness properties) |
| ■ Executable | (implementations generated automatically in 13 languages) |

# 4. Formal Modelling of the MAA as a TRS

# Starting point

- Informal description of the MAA
  - [Davies-Clayden-88] NPL research report
    quasi identical as [ISO standard 8731-2]
  - together with its C implementation
    although incomplete (no "mode of operation")

- Formal description of the MAA
  - [Garavel-Turlier-92] specification in LOTOS and C
  - derived from the LOTOS specification of [Munster-91]

# Outcome

- Formal model of the MAA as a TRS in REC language
- A large model:
  - 46 pages of text (Annex B of our paper)
  - 1575 lines (5 times larger than the largest benchmarks of the Rewrite Engines Competition)
  - 13 sorts
  - 18 constructors
  - 644 non-constructors
  - 684 rewrite rules
    (only 6 conditional rules that can be easily eliminated)

# Good properties

- Our model is exhaustive
  - it describes the full MAA (including "mode of operation")
- Our model is minimal
  - each sort, constructor, and non-constructor defined is actually used (no "dead code")
- Our model is self-contained
  - each detail of the MAA is expressed using TRS only
  - no import of externally-defined types or functions
  - no machine-specific assumptions (e.g., 32-bit vs 64-bit words, big-endian ordering)

# Test vectors

- Cryptographic functions come with test vectors →

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| J | 00FF | 00FF | 00FF | 00FF | 5555 | 5555 | 5555 | 5555 |
| K | 0000 | 0000 | 0000 | 0000 | 5A35 | D667 | 5A35 | D667 |
| P | | FF | | FF | | 00 | | 00 |
| $X_0$ | 4A64 | 5A01 | 4A64 | 5A01 | 34AC | F886 | 34AC | F886 |
| $Y_0$ | 50DE | C930 | 50DE | C930 | 7397 | C9AE | 7397 | C9AE |
| $V_0$ | 5CCA | 3239 | 5CCA | 3239 | 7201 | F4DC | 7201 | F4DC |
| W | FECC | AA6E | FECC | AA6E | 2829 | 040B | 2829 | 040B |
| $M_1$ | 5555 | 5555 | AAAA | AAAA | 0000 | 0000 | FFFF | FFFF |
| X | 48B2 | 04D6 | 6AEB | ACF8 | 2FD7 | 6FFB | 8DC8 | BBDE |
| Y | 5834 | A585 | 9DB1 | 5CF6 | 550D | 91CE | FE4E | 5BDD |
| $M_2$ | AAAA | AAAA | 5555 | 5555 | FFFF | FFFF | 0000 | 0000 |
| X | 4F99 | 8E01 | 270E | EDAF | A70F | C148 | CBC8 | 65BA |
| Y | BE9F | 0917 | B814 | 2629 | 1D10 | D8D3 | 0297 | AF6F |
| S | 51ED | E9C7 | 51ED | E9C7 | 9E2E | 7B36 | 9E2E | 7B36 |
| X | 3449 | 25FC | 2990 | 7CD8 | B1CC | 1CC5 | 3CF3 | A7D2 |
| Y | DB91 | 02B0 | BA92 | DB12 | 29C1 | 485F | 160E | E9B5 |
| T | 24B6 | 6FB5 | 24B6 | 6FB5 | 1364 | 7149 | 1364 | 7149 |
| X | 277B | 4B25 | 28EA | D8B3 | 288F | C786 | D048 | 2465 |
| Y | D636 | 250D | 81D1 | 0CA3 | 9115 | A558 | 7050 | EC5E |
| Z | F14D | 6E28 | A93B | D410 | B99A | 62DE | A018 | C83B |

- Our model is self-checking
  it contains 203 assertions test vectors
  - taken from [Davies-Clayden-88], i.e., [ISO 8731-2]
  - taken from [ISO 8730:1990, Annex E.3.3]
  - added by us, so as to detect:
    - errors arising from byte permutations (endianness issues)
    - incorrect segmentation of messages longer than 256 blocks

# Executability issues

- In principle, TRS encoded in the REC format are executable (by translation to other languages)
- In practice, Peano-style naturals (i.e., in unary notation with zero and succ) exhaust memory
  - the MAA manipulates many blocks (32-bit naturals)
  - blocks cannot be represented in unary notation
  - we represent blocks in binary form (words of 4 octets)
  - logical operations (AND, OR, XOR, CYC) are easy
  - arithmetical operations (ADD, CAR, MUL) are involved

    $\Rightarrow$ 8-bit, 16-bit, and 32-bit adders and multipliers

# Readability

- Our model is readable (despite its size)
  - regular naming conventions for all identifiers
  - constructors chosen appropriately
  - definitions of non-constructors kept simple
- Modular structure:
  - in the MARS repository: the MAA model is a monolithic REC file
  - in Annex B of our paper: the MAA model is split into 21 sections

# Guided tour of the MAA model (1/3)

- 21 sections in Annex B of our paper
- BASIC SORTS
  - 1. Bool sort
  - 2. Nat sort (only used for "small" numbers ≤ 4100)
- MACHINE WORDS
  - 3. Bit sort
  - 4. Octet sort (8 bits)
  - 5. OctetSum sort (9 bits: an Octet and a carry bit)
  - 6. Half sort (16 bits)

# Guided tour of the MAA model (2/3)

- 7. HalfSum sort (17 bits: a Half and a carry bit)
- 8. Block sort (32 bits)
- 9. BlockSum sort (33 bits: a Block and a carry bit)
- 10. Pair sort (64 bits)

## INPUT/OUTPUT DATA

- 11. Key sort (64 bits)
- 12. Message sort (non-empty list of Blocks)
- 13. SegmentedMessage sort (non-empty list of Messages, each containing at most 256 blocks)

# Guided tour of the MAA model (3/3)

- CRYPTOGRAPHIC FUNCTIONS
  - 14. functions CYC, FIX1, FIX2, adjust, PAT, BYT, ADDC
  - 15. functions MUL1, MUL2, MUL2A
  - 16. functions Hi, J1_i, J2_i, K1_i, K2_i
  - 17. Prelude, MainLoop, Coda, Segmentation
- TEST VECTORS
  - 18. Tables 1, 2, and 3 of [Davies-Clayden-88]
  - 19. Table 4 of [Davies-Clayden-88] and other tests
  - 20. Table 5 of [Davies-Clayden-88]
  - 21. Table 6 of [Davies-Clayden-88] and other tests

# 5. Validation of the MAA Model

# Properties

- None of the prior formal MAA specifications (in VDM, Z, and LOTOS) was proven correct

- Our REC specification brings stronger guarantees:
  - confluence
  - termination
  - confluence and termination $\Rightarrow$ all rewrite strategies produce the same result
  - functional correctness of the 203 test vectors

# Confluence and Termination

- Our TRS is deterministic, thus confluent
  - all constructors are free
  - all the rewrite rules that define a non-constructor have disjoint patterns and mutually exclusive premises
  - this was checked by the Opal compiler after automatic translation of the REC model into the Opal language.
- Our TRS is terminating
  - the REC model was automatically translated into the TRS input format of the AProVE tool
  - AProVE produced a proof of quasi-decreasingness (76 steps, 420 pages)

# Functional correctness

- Our REC model was automatically translated into 13 languages: Clean, Haskell, LNT, LOTOS, Maude, mCRL2, OCaml, Opal, Rascal, Scala, Standard ML, Stratego/XT, and Tom
- It was then submitted to 16 tools (compilers, interpreters, and rewrite engines):
  - 11 tools reported that all the 203 test vectors pass
  - (the other tools gave up or timed out)
  - moreover, binary adders and multipliers have been checked separately using 30,000 test vectors

# Two errors detected

- Incorrect test vectors given for function PAT [Davies-Clayden-88, Table 3] and [ISO 8732-2:1992, Table A.3]

```
{X0,Y0}     0103 0703 1D3B 7760     PAT{X0,Y0} EE
{V0,W}      0103 050B 1706 5DBB     PAT{V0,W}  BB
{S,T}       0103 0705 8039 7302     PAT{S,T}   E6
```

should read:

```
{H4,H5}     0000 0003 0000 0060     PAT{H4,H5}  EE
{H6,H7}     0003 0000 0006 0000     PAT{H6,H7}  BB
{H8,H9}     0000 0005 8000 0002     PAT{H8,H9}  E6
```

- Error in the handwritten C function provided to implement the LOTOS function HIGH_MUL
  $\Rightarrow$ mixing formal and non-formal code is risky

# 6. Conclusion

# Contributions

- We revisited the Message Authenticator Algorithm
  - an pioneering algorithm in cryptography (80s)
  - an early application of formal methods (90s)
- We enriched the MARS model repository
  - a formal model of the MAA in the REC language
    - one of the largest handwritten TRS available today
    - self-contained and minimal
    - validated (confluence, termination, test vectors)
  - executable: translations into 13 different languages
  - reusable components (binary adders and multipliers)

# Future work

- Caution! our MAA model is a "tour de force"
  - TRS do not scale well to large problems
  - considerable effort was needed to produce a structured, readable REC model
  - 2-6 times longer than any other (formal or informal) description of the MAA
- Possible uses of our MAA model
  - lab exercises for students (see Annex B.22)
  - assessment of tools (e.g., 1÷140 speed ratio)
  - provers: verify correctness of binary adders/multipliers