Formal Methods within Certification Programs Workshop 2024, USA

AI-ASSISTED FORMAL METHOD VERIFICATIONS ON CRYPTOGRAPHIC DESIGNS AND IMPLEMENTATIONS

Long Ngo

Teron Labs, Australia

TERON LABS

AGENDA

- A quick look at Formal Methods used in Cryptography
- Examples of Verification Tools for cryptography
  - Formal Verifiers with non or limited smart assistance
  - AI-based verifiers
- How AI can help Formal Verification
- Challenges and Considerations
  - Gaps
  - How will it fit in certifications?
  - Our R&D work
- Wrap up and Q&A

PART 1: FORMAL METHODS IN CRYPTOGRAPHY

VS

TWO SCHOOLS OF CRYPTOGRAPHIC ANALYSIS

Computational models

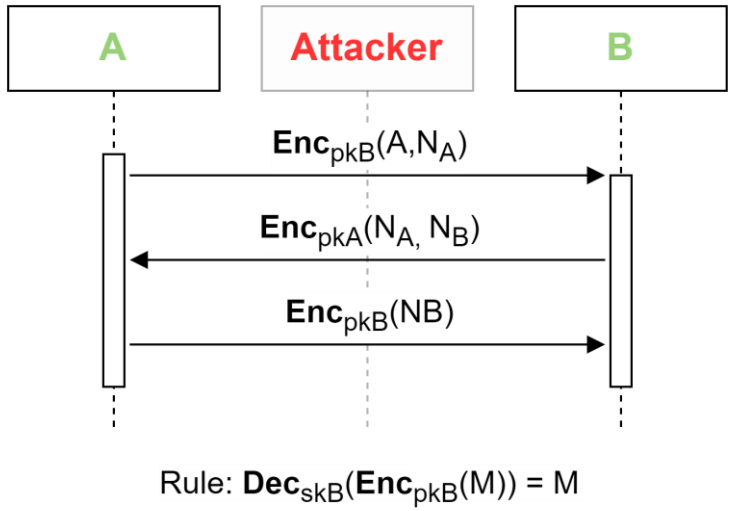
- **Honest parties** follow probabilistic/computational algorithms
- **Attacker:** Computational/Probabilistic
- Probabilistic **security definitions:** security ≈ attacking not better than guessing

GAP = analysis results are not equivalent

Symbolic models

(formal method-based models)

- **Honest parties** follow symbolic representations of the algorithms
- **Attacker**: plays symbolic rules
- Symbolic **security definitions:** security ≈ secret symbols not revealed



TERON LABS

FORMAL METHODS IN CRYPTOGRAPHY

Formal specification

- Propositional logic: *"AES is secure"*
- Modal logic: *A believes key(K_{AB}, A↔B)*
- First-order term algebra

$$
\begin{array}{lll}
t ::= & & \text{term} \\
\mid & x & \text{variable } x \\
\mid & a & \text{name } a \\
\mid & f(a) & \text{application of symbol } f \in \{\mathsf{pub}, \mathsf{priv}\} \text{ on a name} \\
\mid & f(t_1, t_2) & \text{application of symbol } f \in \{\mathsf{enc}, \mathsf{enca}, \mathsf{sign}, \langle\,\rangle\}
\end{array}
$$

- Formal semantics: denotational, operational, axiomatic ones and so on

Formal reasoning

- Logical rules: modus ponens, universal instantiation and so on :

If $P \to Q$ and P, then Q. If $\forall x\, P(x)$, then $P(a)$.

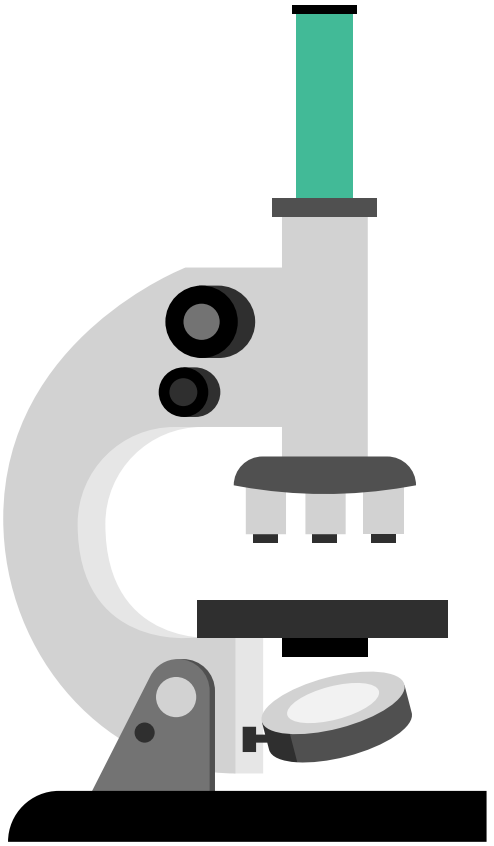
- Cryptographic rules

$$\frac{\mathsf{enc}(x, \mathsf{k}(y)) \qquad \mathsf{k}(y)}{x}$$

Formal analysing

- Manual
- **Automated tools**
  - **Theorem proving: Isabelle/HOL, Coq …**
  - **Model checking: Scyther, Proverif, CryptoVerif …**

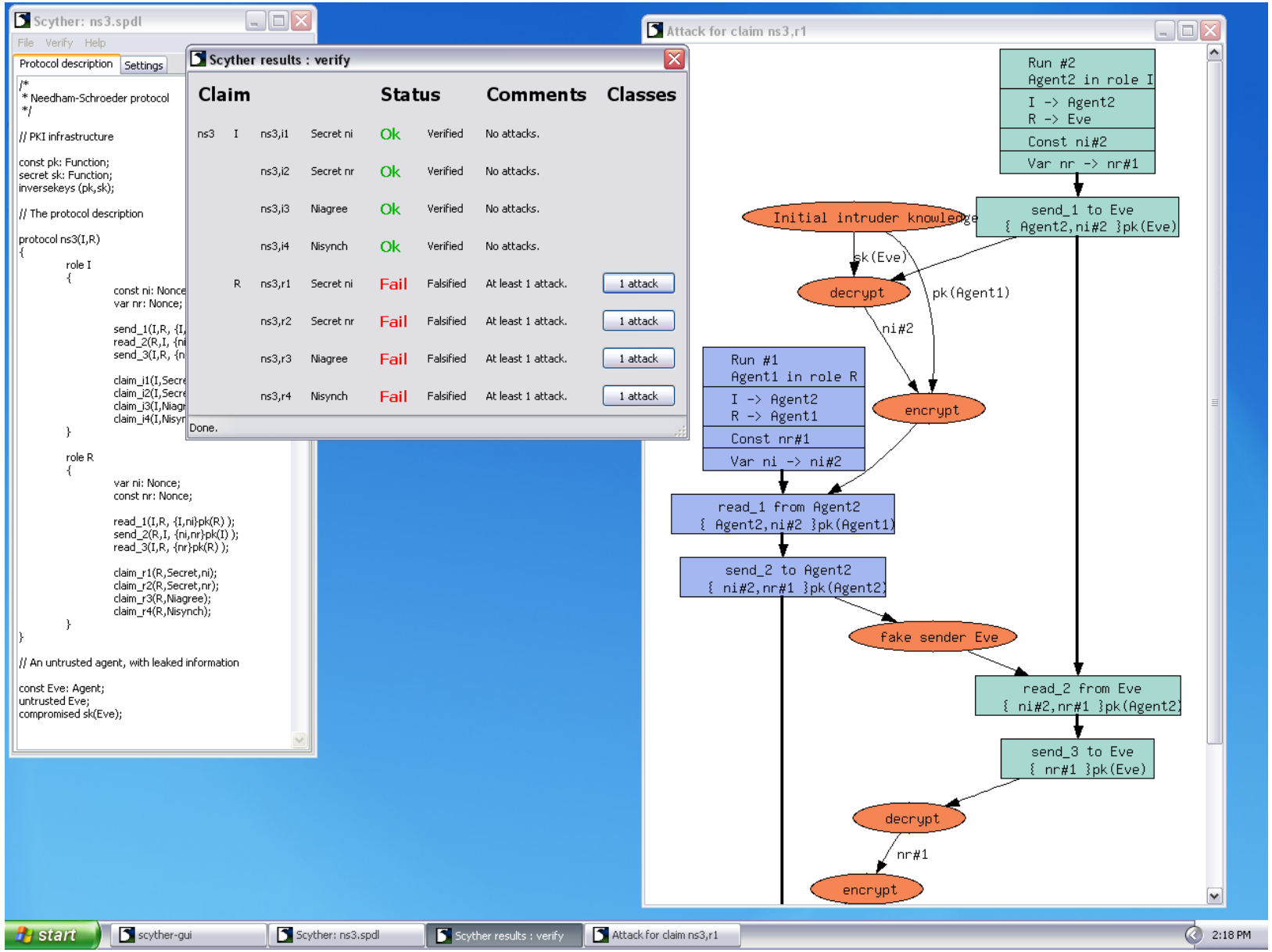
PART 2:EXAMPLES



EXAMPLE 1

SCYTHER

SYMBOLIC MODEL CHECKER



TERON LABS

BRIDGING THAT GAP

- We want the best of the both worlds
  - Formal and mechanizable design and verification
  - Computationally sound
- Indirect way:
  - Use a symbolic model
  - Prove that symbolic security implies computational security
- Direct way
  - Use a formal model with computational semantics.
  - Prove computational security directly

EXAMPLE 2

CRYPTOVERIF

COMPUTATIONAL MODEL CHECKER

```
Cryptoverif. Cryptographic protocol verifier, by Bruno Blanchet
Copyright ENS-CNRS, distributed under the CeCILL-B license
  -lib <filename>       choose library file
  -tex <filename>       choose TeX output file
  -in channels / -in oracles    choose the front-end
  -impl         get implementation of defined modules
  -o <directory>        if "-impl" is given, the generated files will be placed
in this directory (Default: .)
  -help  Display this list of options
  --help  Display this list of options
```

```
Proved event(simu_forgery) ==> true in game 33
Adv[Game 1: event(simu_forgery) ==> true] <= NS * max(4 * NS, 1) * max(4 * NS, 1
) * pCDH(time(context for game 28) + time + (2 * NS + 1 + NH) * time(exp)) + (2
* NS + 2 * NH) / |Z| + (-1 * qD + -1 * qE + qD * qD + 2 * qD * qE + qE * qE + NS
) / |G| + Adv[Game 33: event(simu_forgery) ==> true]
Adv[Game 33: event(simu_forgery) ==> true] <= 0
RESULT Proved event(simu_forgery) ==> true up to probability NS * max(4 * NS, 1)
 * max(4 * NS, 1) * pCDH(time(context for game 28) + time + (2 * NS + 1 + NH) *
time(exp)) + (2 * NS + 2 * NH) / |Z| + (-1 * qD + -1 * qE + qD * qD + 2 * qD * q
E + qE * qE + NS) / |G|
RESULT time(context for game 28) = NS * NS * time(= bitstring, maxlength(game 28
: m), maxlength(game 28: m)) + NH * NS * time(= bitstring, maxlength(game 28: m)
, maxlength(game 28: x_1)) + (NS + 4 * NU) * time(exp) + (NS + 2 * NU) * time(mu
lt) + NS * NS * time(= bitstring, maxlength(game 28: m'), maxlength(game 28: m))
 + NH * NS * time(= bitstring, maxlength(game 28: m'), maxlength(game 28: x_1))
+ NS * NH * time(= bitstring, maxlength(game 28: x_1), maxlength(game 28: m)) +
NH * NH * time(= bitstring, maxlength(game 28: x_1), maxlength(game 28: x_1)) +
qH1 * time(Hash1, maxlength(game 28: x1)) + qH2 * time(Hash2, maxlength(game 28:
 x2)) + NU * time([1,NU]) + NU * time(pair_G2) + NU * time(changetype) + NU *
time(pair_S) + NU * time(pair_3) + NU * time(changetype1) + NU * time(plus_4) +
NU * time(add_1) + NU * time(plus) + NU * time(Hash2, maxlength(game 28: mv)) +
NU * time(pair_W)
All queries proved.
```

EXAMPLE 3

SOFTWARE ANALYSIS WORKBENCH

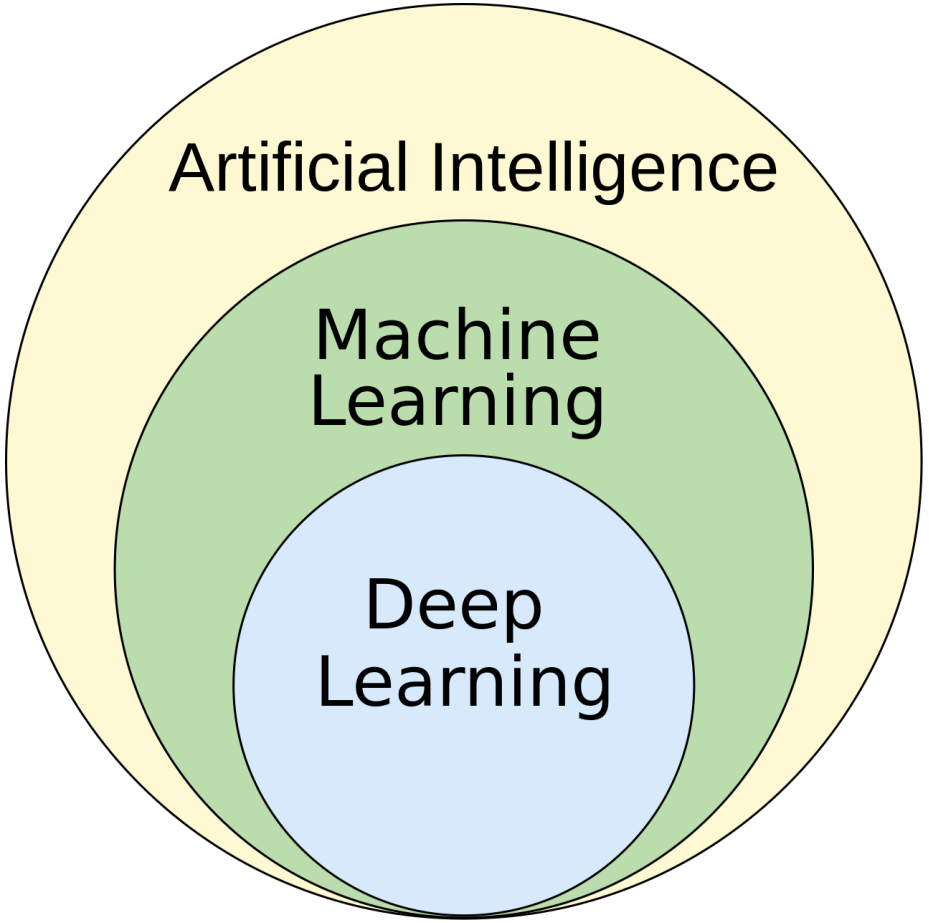
REAL CODE EQUIVALENCE PROVER

```c
// multiply_standard
//
// Multiply two 32-bit integers in a standard way, cast each to a local 32-bit
// number and then multiply and return the result.
uint32_t multiply_standard(uint16_t a, uint16_t b) {
  uint32_t local_a = (uint32_t) a;
  uint32_t local_b = (uint32_t) b;
  uint32_t result;

  result = local_a * local_b;

  return result;
}
```

```c
uint32_t multiply_textbook(uint16_t a, uint16_t b) {
  uint8_t a_1 = (uint8_t)( (0xff00 & a) >> 8 );
  uint8_t a_0 = (uint8_t)( (0x00ff & a) >> 0 );
  uint8_t b_1 = (uint8_t)( (0xff00 & b) >> 8 );
  uint8_t b_0 = (uint8_t)( (0x00ff & b) >> 0 );

  uint16_t z0 = (uint16_t)a_0 * (uint16_t)b_0;
  uint16_t z1 = (uint16_t)a_1 * (uint16_t)b_0;
  uint16_t z2 = (uint16_t)a_0 * (uint16_t)b_1;
  uint16_t z3 = (uint16_t)a_1 * (uint16_t)b_1;

  uint32_t result = 0;
  result += (uint32_t)z0;
  result += (uint32_t)z1 << 8;
  result += (uint32_t)z2 << 8;
  result += (uint32_t)z3 << 16;

  return result;
}
```

```
[17:32:52.661] Extracting reference term: multiply_standard
[17:32:52.696] Extracting implementation term: multiply_textbook
[17:32:52.699] Extracting implementation term: multiply_karatsuba
[17:32:52.702] Proving equivalence: multiply_standard == multiply_textbook
[22:14:04.689] Valid
[22:14:04.697] Proving equivalence: multiply_standard == multiply_karatsuba
[04:16:05.144] Valid
[04:16:05.146] Done.
```

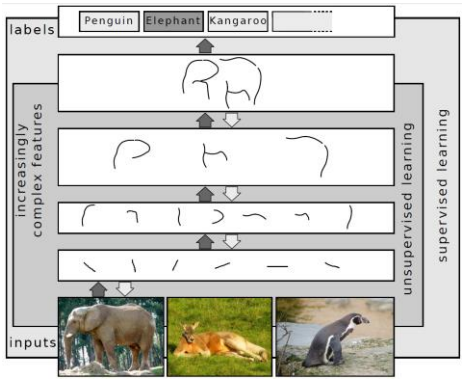
PART 3: HOW AI CAN HELP



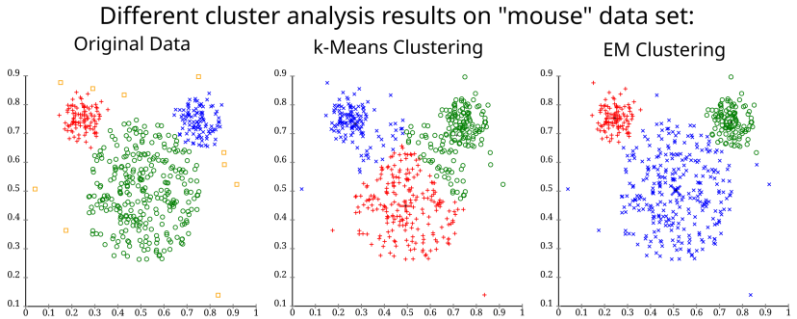
TERON LABS

MACHINE LEARNING WORLD

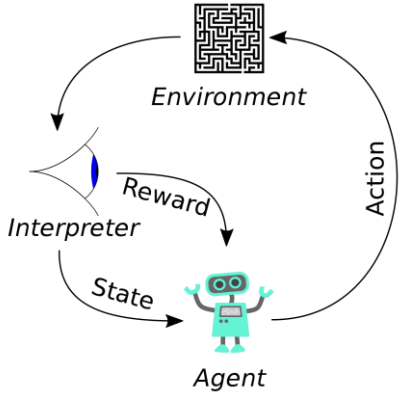
- Supervised Learning
  - Trained on labelled data
  - Predict the label of new data



- Unsupervised Learning
  - Trained on unlabelled data
  - Find patterns.



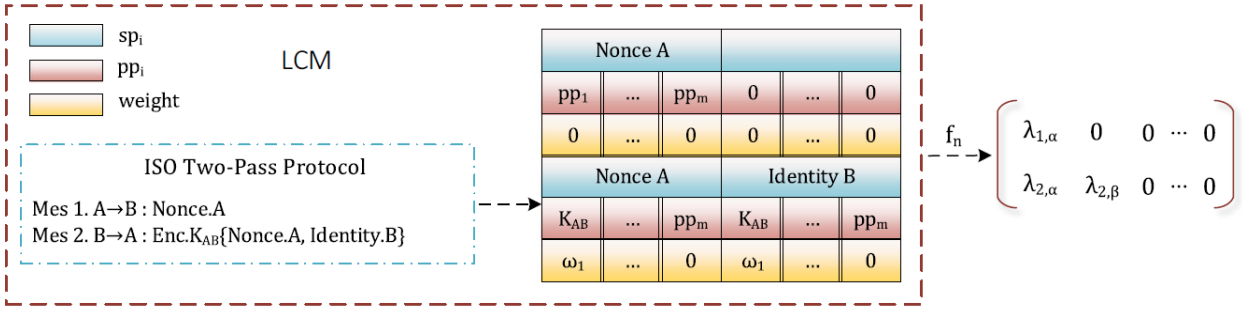
- Reinforcement Learning
  - Interact with environments.
  - Make decisions.

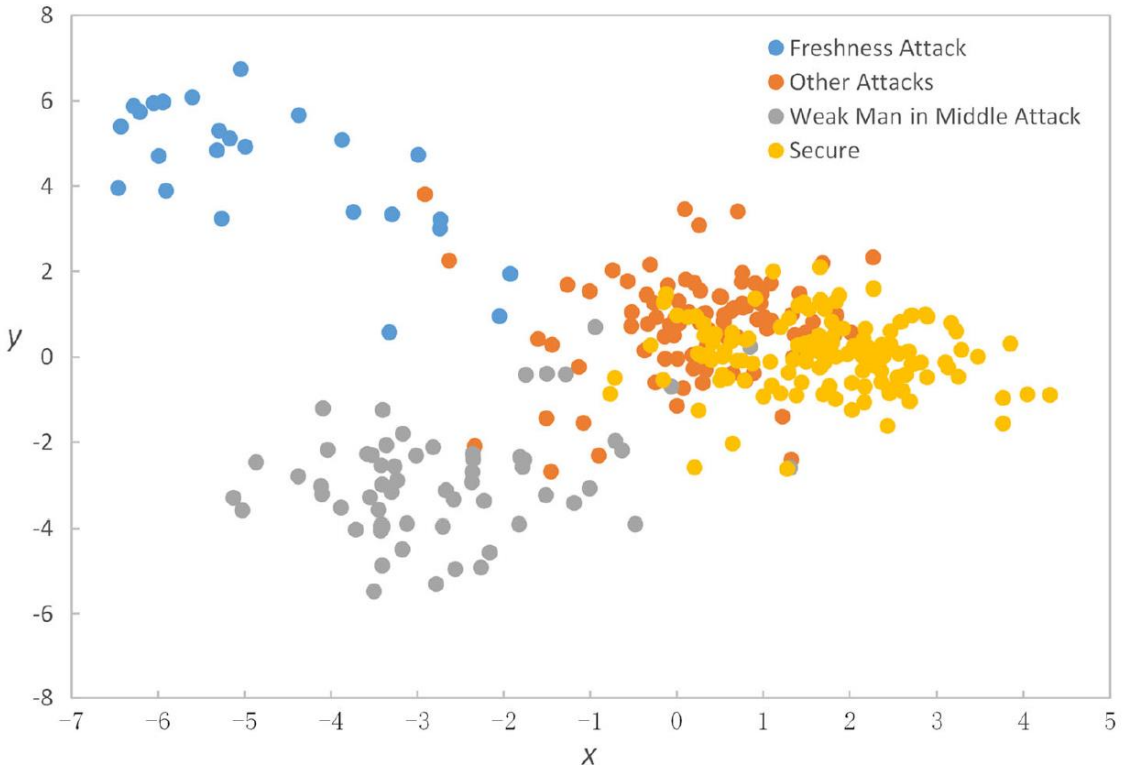


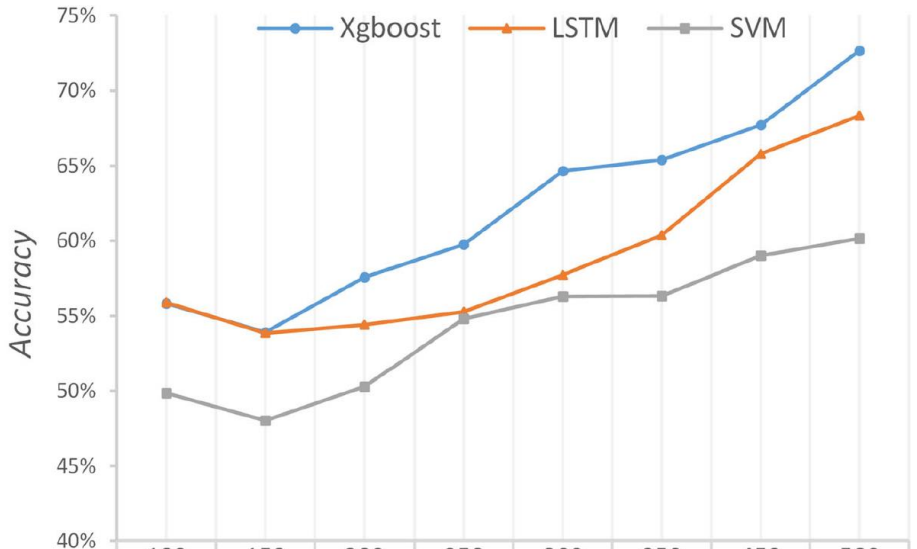
TERON LABS

EXAMPLE 4: MACHINE-LEARNING BASED

from *A machine learning-based scheme for the security analysis of authentication and key agreement protocols by Ma et al, 2018*







	100	150	200	250	300	350	450	500
Xgboost	0.5582	0.5391	0.5758	0.5973	0.6466	0.6539	0.6772	0.7267
LSTM	0.5588	0.5385	0.5441	0.5526	0.5773	0.6036	0.6578	0.6834
SVM	0.4985	0.4802	0.5028	0.548	0.563	0.5633	0.5899	0.6013

CHALLENGES AND CONSIDERATIONS

HOW IS **AI CHANGING** OUR FIELD?

- **Old gap** between the formal method and cryptography worlds is narrowed
- **New gap** appears between the AI-based verifiers and the rest
- AI-dominant methods would be good to find what can be “**potentially**” insecure or secure
  - Anomalies detection
  - Preliminary classifications.
- AI-assisted methods could help existing formal methods used by cryptographers
  - Strategy selection for theorem prover
  - State space reduction for model checker
  - Formal specification mining.

HOW WILL IT FIT IN CERTIFICATIONS?

- **FIPS 140-3**:
  - Known Answer Test (KAT) vectors are **practically** sufficient.
  - Beneficial to code review

- **Common Criteria**: there are different levels of assurance.
  - High assurance functional testing
    - Static analysis
    - Implementation correctness
  - Thorough vulnerability finding
    - Side-channel attacks
    - Smart fuzzing
    - Resource leak

Assurance Class	Assurance components
ADV: Development	ADV_ARC.1 Security architecture description
	ADV_FSP.6 Complete semi-formal functional specification with additional formal specification
	ADV_IMP.2 Complete mapping of the implementation representation of the TSF
	ADV_INT.3 Minimally complex internals
	ADV_SPM.1 Formal TOE security policy model
	ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.5 Advanced support
	ALC_CMS.5 Development tools CM coverage
	ALC_DEL.1 Delivery procedures
	ALC_DVS.2 Sufficiency of security measures
	ALC_LCD.2 Measurable life-cycle model
	ALC_TAT.3 Compliance with implementation standards - all parts
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	ASE_INT.1 ST introduction
	ASE_OBJ.2 Security objectives
	ASE_REQ.2 Derived security requirements
	ASE_SPD.1 Security problem definition
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.3 Rigorous analysis of coverage
	ATE_DPT.4 Testing: implementation representation
	ATE_FUN.2 Ordered functional testing
	ATE_IND.3 Independent testing - complete
AVA: Vulnerability assessment	AVA_VAN.5 Advanced methodical vulnerability analysis

Table 8 - EAL7

A TOY EXAMPLE

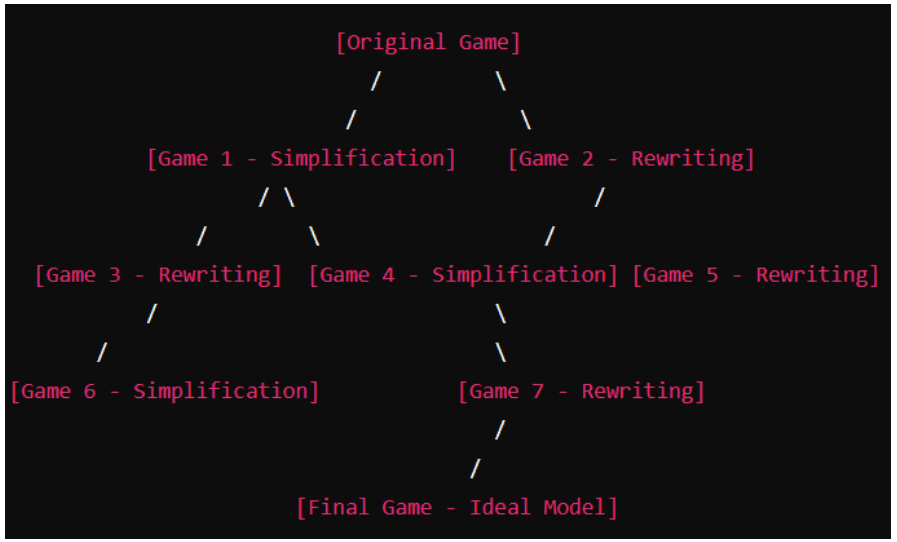
- Would Known Answer Tests be enough?
- Would ChatGPT help?

```python
# First Implementation

def hash(input):
    temp = compute_sha2_256(input)
    if temp < 1000:
        crash()
    else:
        return temp
```

```python
# Second Implementation

def hash(input):
    temp = compute_sha2_256(input)
     return temp
```

OUR R&D DIRECTIONS

- AI-assisted verification cryptographic designs



- AI-assisted implementation equivalence verification.

```python
# Program 1
def caesar_cipher(text, shift):
    return ''.join(
        chr(
            (ord(char) - (65 if char.isupper() else 97) + shift) % 26 +
            (65 if char.isupper() else 97)
        )
        if char.isalpha() else char
        for char in text
    )
```

≡ ?

```python
# Program 2
def caesar_cipher(text, shift):
    encrypted = ""
    for char in text:
        if char.isalpha():
            shift_amount = 65 if char.isupper() else 97
            encrypted += chr(
                (ord(char) - shift_amount + shift) % 26 + shift_amount
            )
        else:
            encrypted += char
    return encrypted
```


TERON LABS

SUMMARY + Q&A + THANK YOU

- Formal methods have been used for cryptography in
  - Designing
  - Verifying
  - Implementing
- Artificial Intelligence is helping Formal methods
- Any questions now?
- Or later, email me at **long@teronlabs.com**.