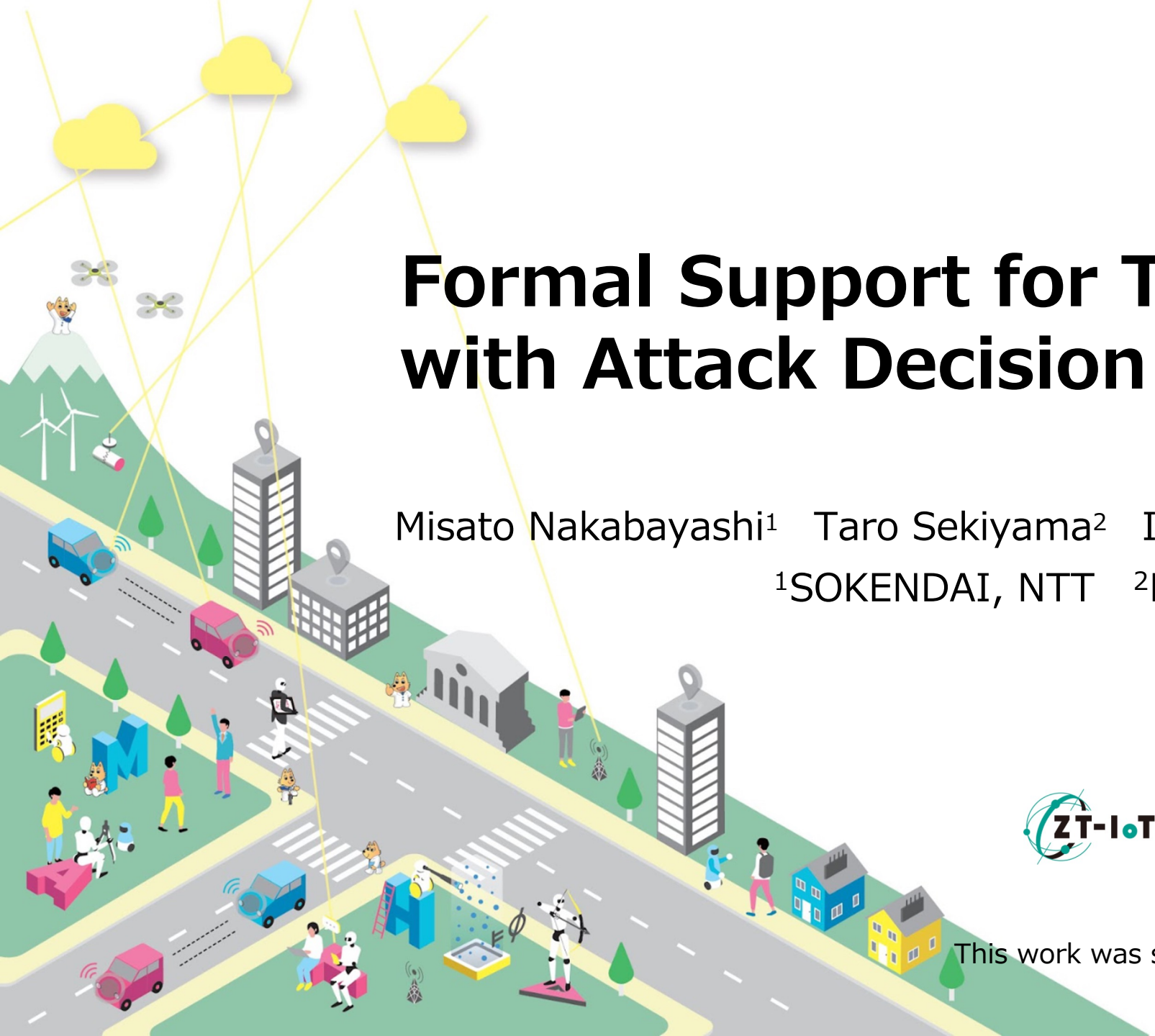


Formal Support for Threat Modeling with Attack Decision Diagrams

Misato Nakabayashi[1] Taro Sekiyama[2] Ichiro Hasuo[2] Yutaka Ishikawa[2]

[1]SOKENDAI, NTT [2]National Institute of Informatics

This work was supported by JST CREST JPMJCR21M3, Japan

Background

- Security vulnerabilities in the system can cause serious damage, such as leakage of confidential information or unauthorized use of equipment, and
- Once deployed, a system cannot be easily modified

- Important to conduct threat analysis at the design phase of the system to understand and manage security risks
  - **However, exhaustive threat analysis is difficult**

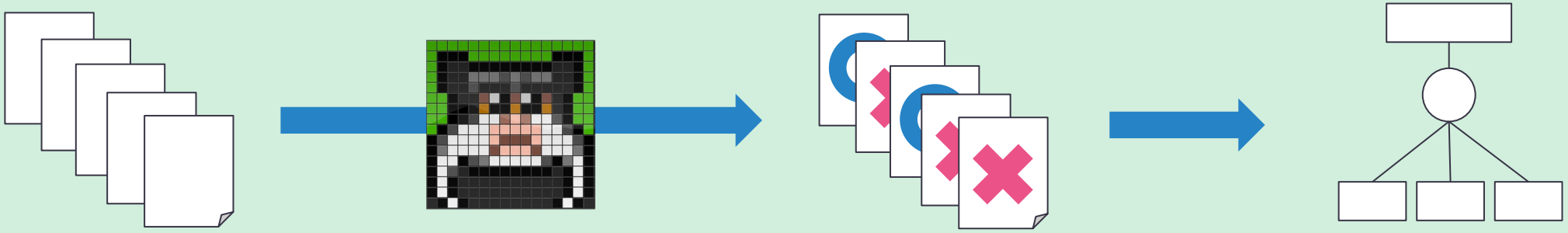
Background

Threat analysis is conducted by four steps:

1. Determine assets to be protected,
2. Identify threats against the assets,
3. Identify attack methods rousing the threats to organize the risks caused by the identified threats, and
4. Assessment risk: score the identified risks to prioritize countermeasures

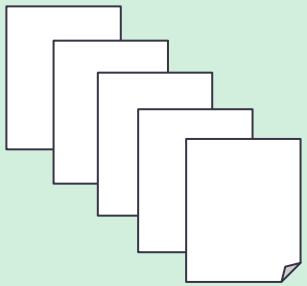
Contribution

We propose the Formal Support for Threat Modeling (FSTM) system, a method to visualize threats to support threat modeling using a formal verification tool

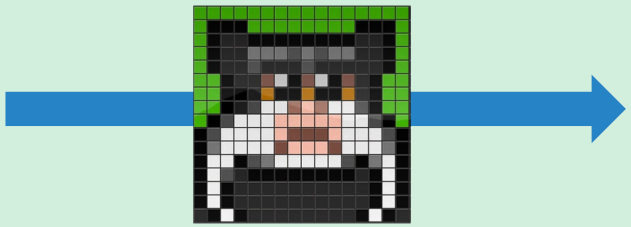


- Exhaustive verification requires verification of all attack combination patterns, but the number of verification times can be reduced by considering the monotonicity of security
- Automatic generation of codes for each attack combination from a single verification code
- Outputs verification results in an AND-OR tree format that shows the causal relationship between the attacker's behavior and security

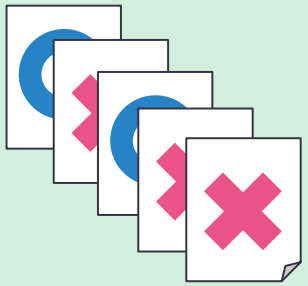
Proposed Method Overview



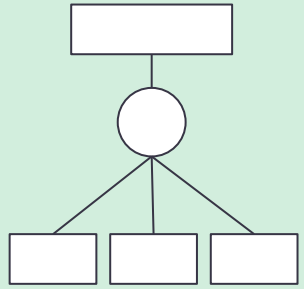
Set of
Attack Patterns



Verification using
Formal Verification



Set of Attack Patterns
with Security Labels



Attack Decision Diagram (ADD)

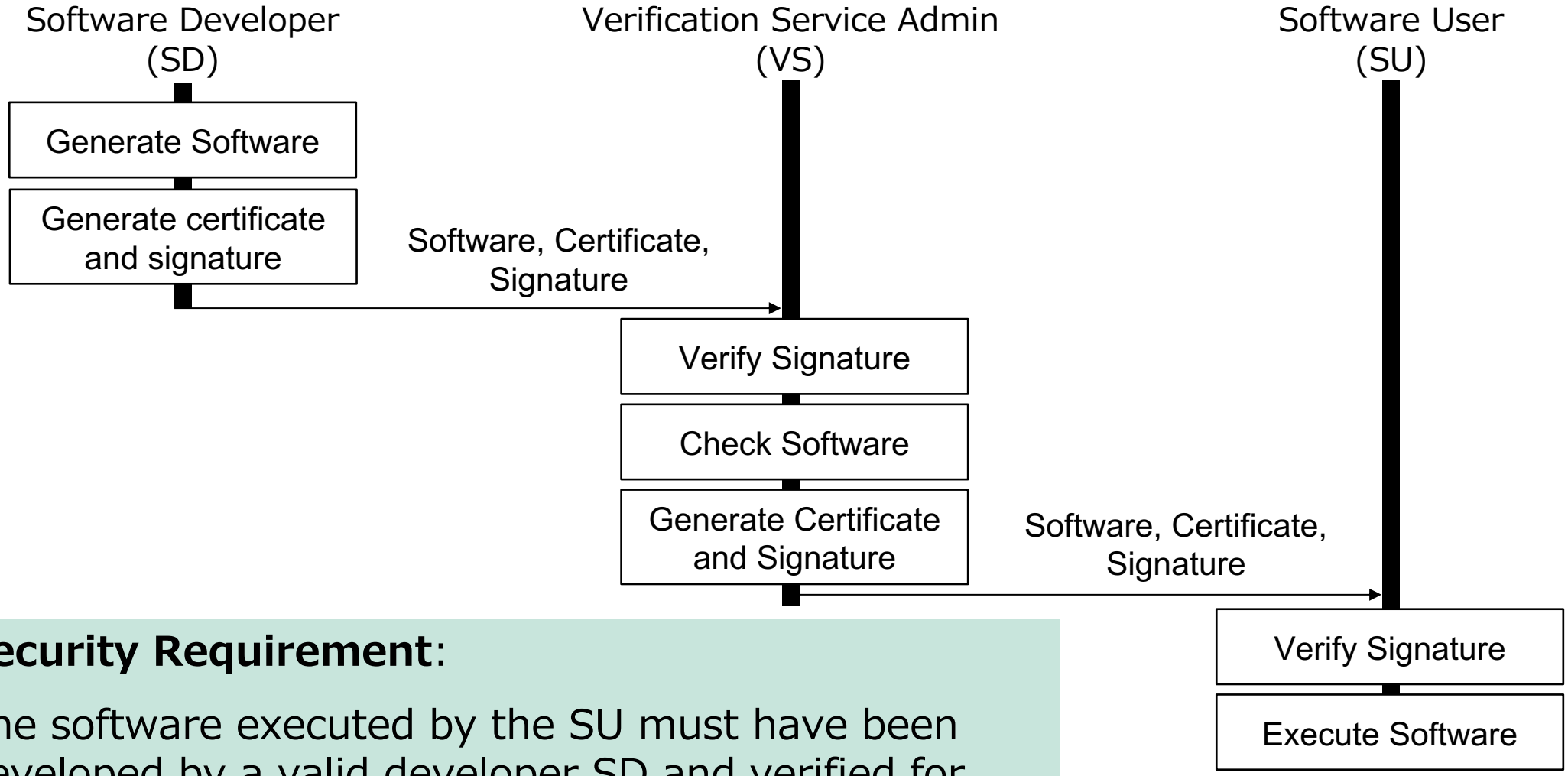
1 Determine attack elements based on assumed threats and create attack patterns

2 Reduce the number of verifications using monotonicity of security

3 Create attack decision diagrams (ADD) using verification results

Proposed Method Case Study

Software Authentication System



Security Requirement:

The software executed by the SU must have been developed by a valid developer SD and verified for vulnerabilities by VS

Proposed Method

1 Generate attack patterns based on assumed threats

Assumed Threats

- Leakage of secret information
- Tampering with messages in communication channels
- Eavesdropping on messages in communication channels

We call each element **attack component**

- a_1: Reveal SD's secret key for signing, a_2: Reveal VS's secret key for signing
- a_3: Tamper message 1, a_4: Tamper message 2
- a_5: Eavesdrop message 1, a_6: Eavesdrop message 2

We call the combination of attack components **attack pattern**

- p_1: $a_1 \wedge a_2$, p_2: $a_1 \wedge a_3$, ... (64 patterns)

Proposed Method

2 Reducing the Number of Verification Using Monotonicity

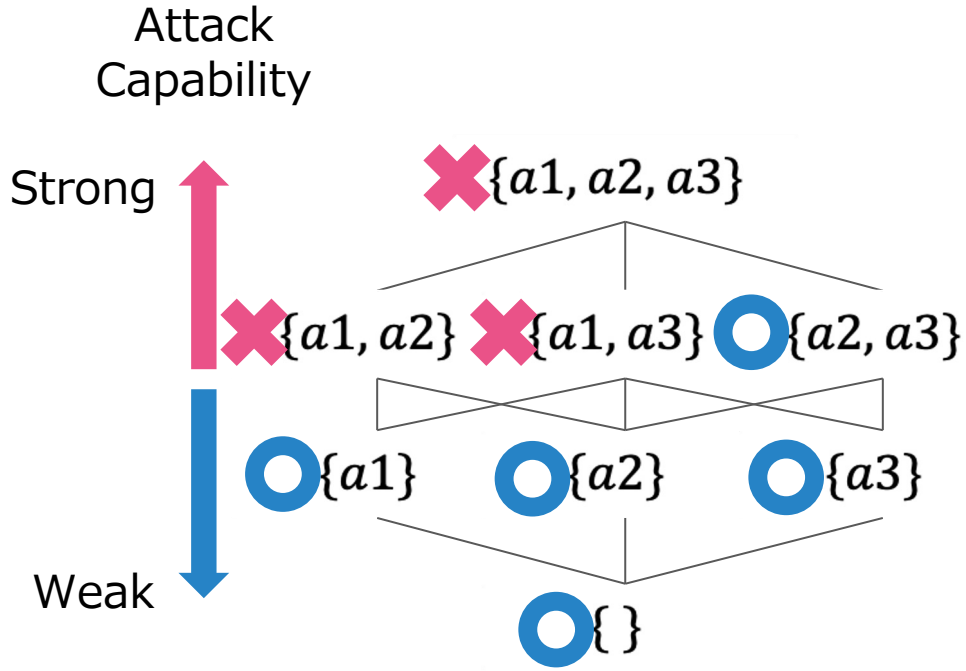
Monotonicity of Security

For attack patterns p_1 and p_2 with $p_1 \leq p_2$, if a system is insecure on p_1, then it is also insecure on p_2

- E.g., if a system is insecure on "a_1: Reveal SD's secret key for signing", it is also insecure on "a_1: Reveal SD's secret key for signing $\wedge$ a_2: Reveal VS's secret key for signing"

Proposed Method

2 Reducing the Number of Verification Using Monotonicity



: insecure

: secure

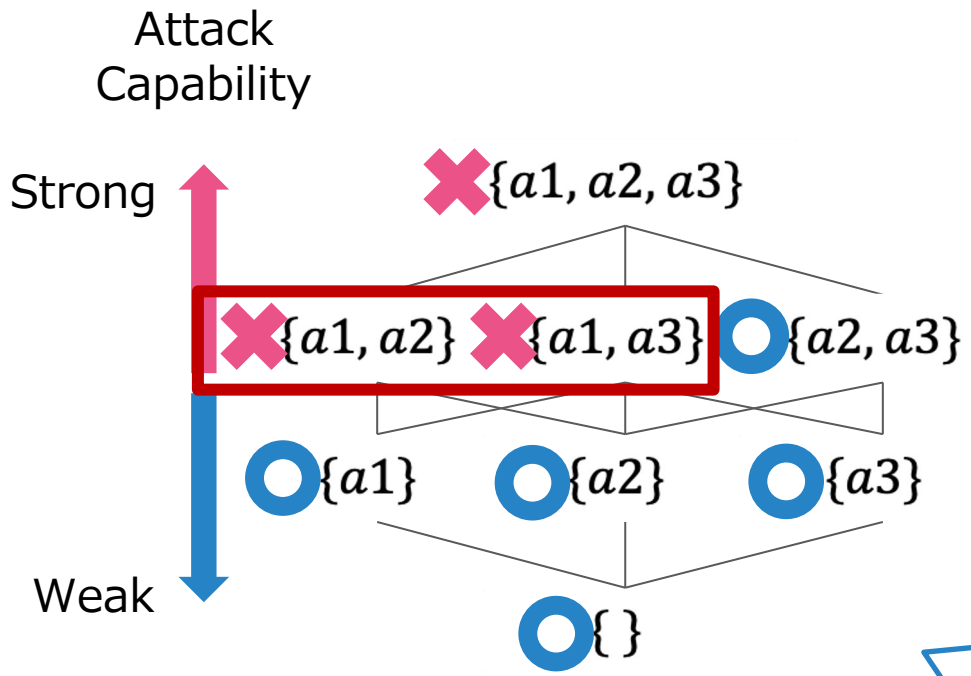
Exhaustive Verification
II
Labelling all attack patterns as secure or insecure

Verification times can be reduced using inference based on monotonicity

In the case study,
64 times → **20 times**

Proposed Method

3 Generating ADDs using Verification Results

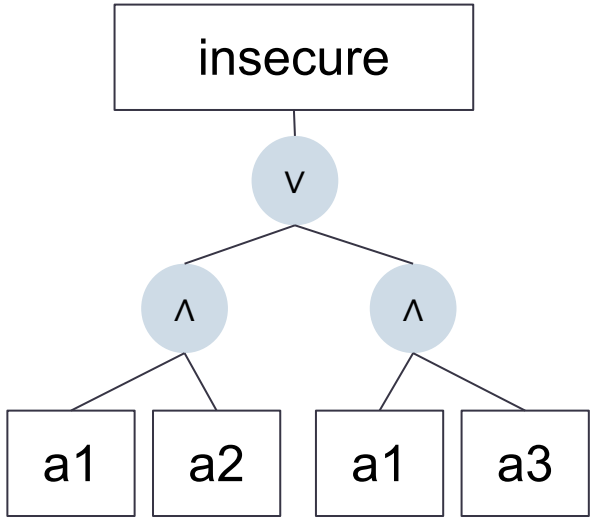


: insecure

: secure

Represents Verification Results in DNF-Format

Find the minimal attack patterns that are insecure and connect them with logical sums



Proposed Method

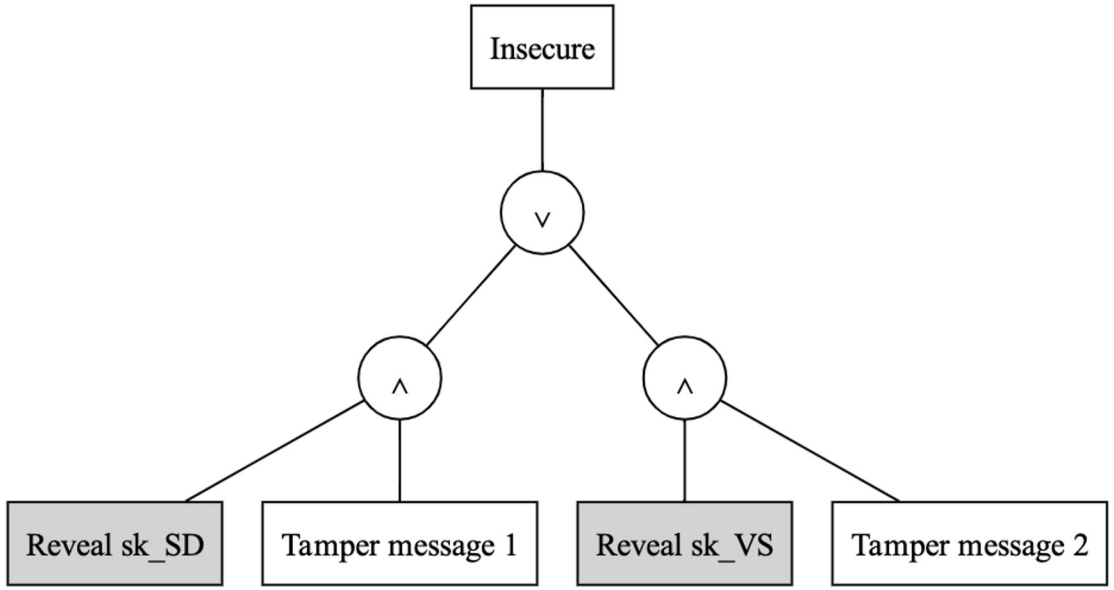
3 Generating ADDs using Verification Results

Verification Results and ADD of case study

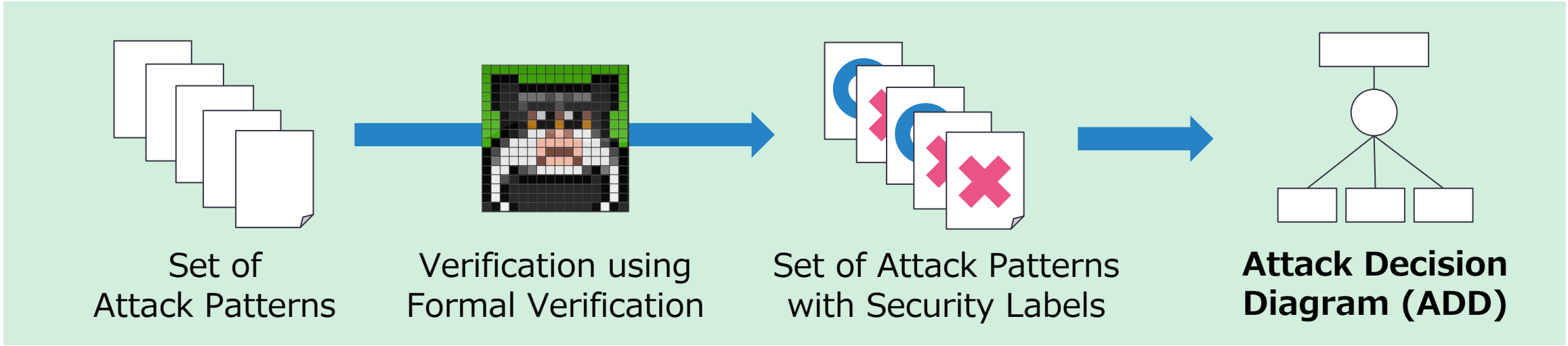
	Reveal sk_{SD}	Reveal sk_{VS}	Tamper m_1	Tamper m_2	Eavesdrop m_1	Eavesdrop m_2	Verification result
0	0	0	0	0	0	0	Verified
1	0	0	0	0	0	1	Verified
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
19	0	1	0	0	1	1	Verified
20	0	1	0	1	0	0	Falsified
21	0	1	0	1	0	1	Falsified
22	0	1	0	1	1	0	Falsified
23	0	1	0	1	1	1	Falsified
24	0	1	1	0	0	0	Verified
25	0	1	1	0	0	1	Verified
26	0	1	1	0	1	0	Verified
27	0	1	1	0	1	1	Verified
28	0	1	1	1	0	0	Falsified
29	0	1	1	1	0	1	Falsified
30	0	1	1	1	1	0	Falsified
31	0	1	1	1	1	1	Falsified
32	1	0	0	0	0	0	Verified
33	1	0	0	0	0	1	Verified
34	1	0	0	0	1	0	Verified
35	1	0	0	0	1	1	Verified
36	1	0	0	1	0	0	Verified
37	1	0	0	1	0	1	Verified
38	1	0	0	1	1	0	Verified
39	1	0	0	1	1	1	Verified
40	1	0	1	0	0	0	Falsified
41	1	0	1	0	0	1	Falsified
42	1	0	1	0	1	0	Falsified
43	1	0	1	0	1	1	Falsified
44	1	0	1	1	0	0	Falsified
45	1	0	1	1	0	1	Falsified
46	1	0	1	1	1	0	Falsified
47	1	0	1	1	1	1	Falsified
48	1	1	0	0	0	0	Verified
49	1	1	0	0	0	1	Verified
50	1	1	0	0	1	0	Verified
51	1	1	0	0	1	1	Verified
52	1	1	0	1	0	0	Falsified
53	1	1	0	1	0	1	Falsified
54	1	1	0	1	1	0	Falsified
55	1	1	0	1	1	1	Falsified
56	1	1	1	0	0	0	Falsified
57	1	1	1	0	0	1	Falsified
58	1	1	1	0	1	0	Falsified
59	1	1	1	0	1	1	Falsified
60	1	1	1	1	0	0	Falsified
61	1	1	1	1	0	1	Falsified
62	1	1	1	1	1	0	Falsified
63	1	1	1	1	1	1	Falsified

Security Requirement:

The software executed by the SU must have been developed by a valid developer SD and verified for vulnerabilities by VS



Implementation Using Tamarin Prover

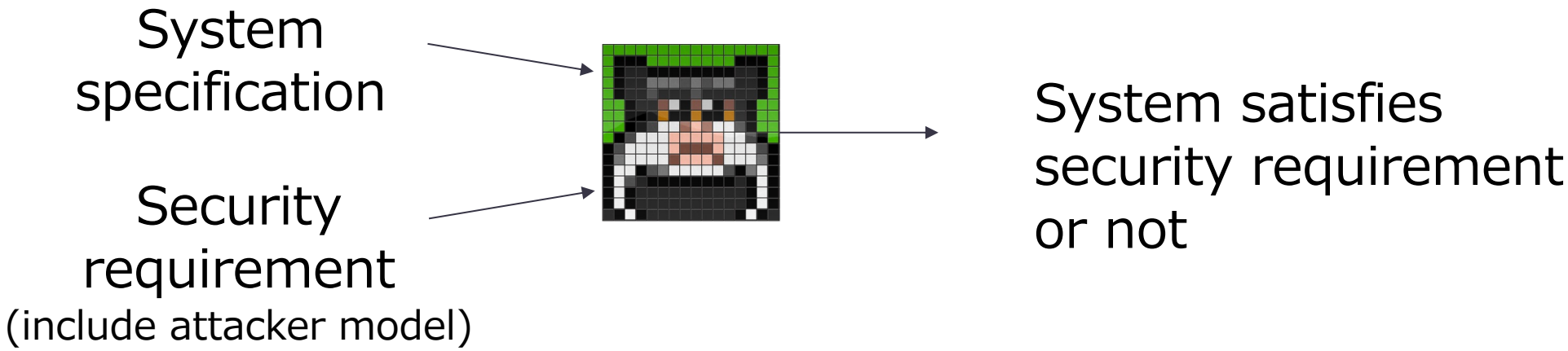


- To perform exhaustive verification, verification codes corresponding to each attack pattern are required
- Consider how to automatically generate verification codes corresponding to each attack pattern from the original code

Implementation Using Tamarin Prover

Tamarin Prover

- Formal Verification Tool for Security Systems

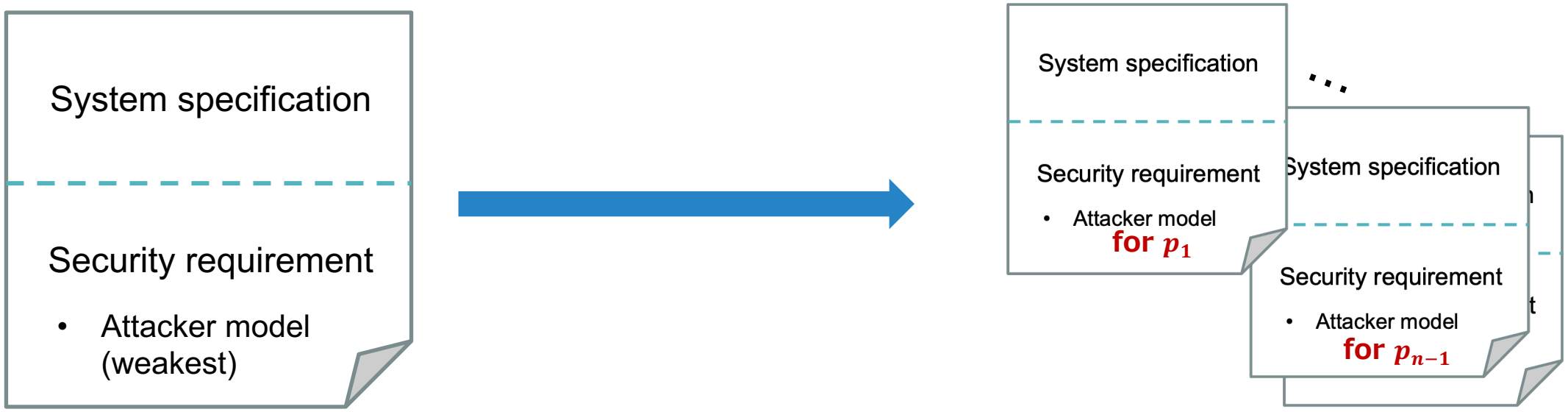


- System specifications are described using multiset rewriting rules and security requirements are described using first-order logic formulas

Implementation Using Tamarin Prover

Generate Code for Each Attack Pattern

- Automatic generation of Tamarin code for each attack pattern from Tamarin code for the weakest attack pattern



Original Tamarin Code

Tamarin Code for Each Attack Pattern

Summary

- We propose a threat visualization method based on exhaustive verification using formal verification tools
- This tool can be used to discover threats that have not appeared in informal threat analysis, to assist in generating attack trees, and to confirm the correctness of threat modeling

Future Issues

- Extension of Targeted Threats
- Improvement of ADD format