29TH USENIX SECURITY SYMPOSIUM

SmartVerif: Push the Limit of Automation Capability of Verifying Security Protocols by Dynamic Strategies

Yan Xiong, Cheng Su, Wenchao Huang*, Fuyou Miao, Wansen Wang, Hengyi Ouyang

University of Science and Technology of China

Reporter: Cheng Su

Background

- The design of security protocols is error-prone
  - 4G (Shaik et al, NDSS, 2016)
  - TLS (Cremers et al, CCS, 2017)
  - 5G (Basin et al, CCS, 2018)
  - …

LILY HAY NEWMAN SECURITY 11.12.2019 11:30 AM

As 5G Rolls Out, Troubling New Security Flaws Emerge

Researchers have identified 11 new vulnerabilities in 5G—with time running out to fix them.

Attacks on TLS vulnerabilities: Heartbleed and beyond

Posted by Mantej Singh Rajpal on Wednesday, November 29th, 2017

Background

- Verifying security protocols has become a key issue in security
- Current related approaches

	Formal Methods	Other Approaches (e.g. fuzzing)
Pros	Formal security guarantee	Full automation
Cons	Partial automation	False positive & False negative

- Motivation: achieve full automation in formal methods

Observation

- State-of-the-art formal method tools

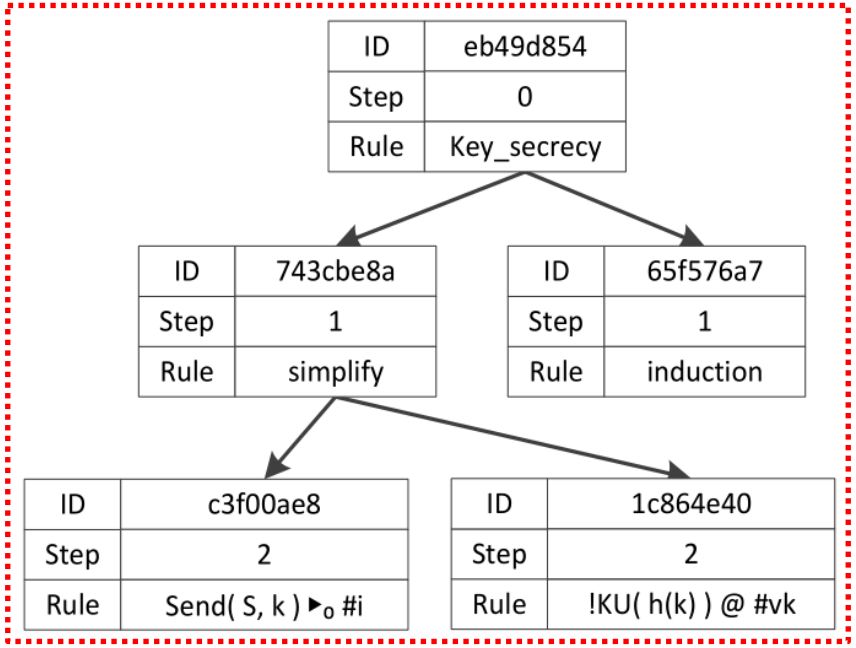
Use static strategies	Loose automation capability
• Achieve automated verification for some protocols • Cannot verify complicated protocols (e.g. Yubikey) • ProVerif, StatVerif, Set-pi, …	• Require human guidance • Tamarin Prover, GSVerif, …

Our Work

- We implement SmartVerif
  - Push the limit of automation capability of verifying security protocols
    - Full automation: no human guidance
    - Generality: 100% success rate in verifying 24 studied protocols (including Yubikey)
  - Achieved by designing a novel dynamic strategy
    - Replacing the original strategies in Tamarin Prover

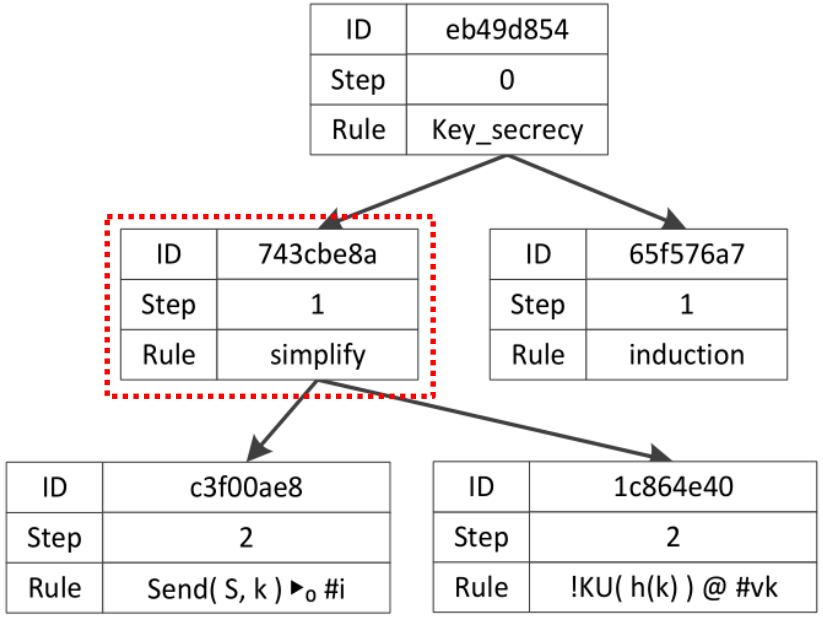
Problem Definition

- Verification process in our work
  - Verification tree



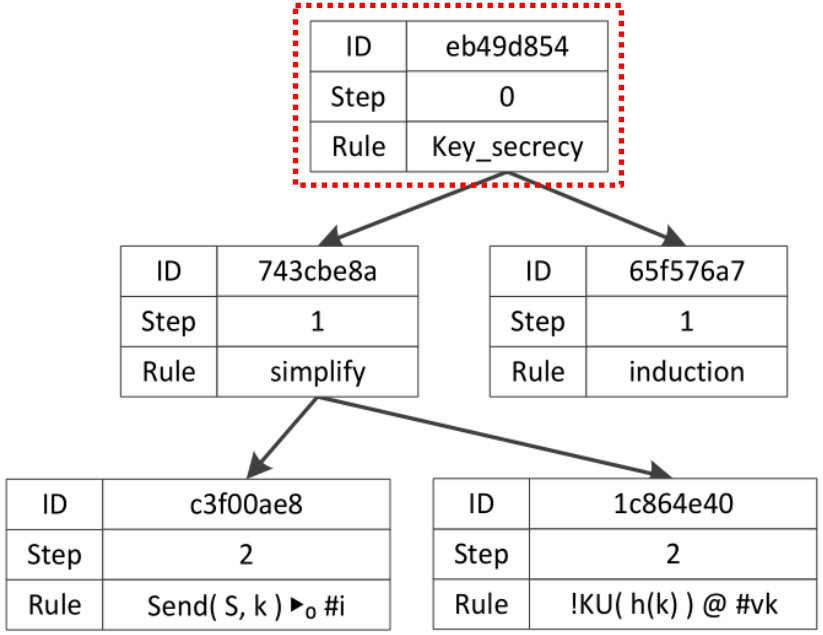
Problem Definition

- Each node in the tree represents a proof state
  - Rule: a proposition
  - Step: the proof step number
  - ID: the hash of the proposition



Problem Definition

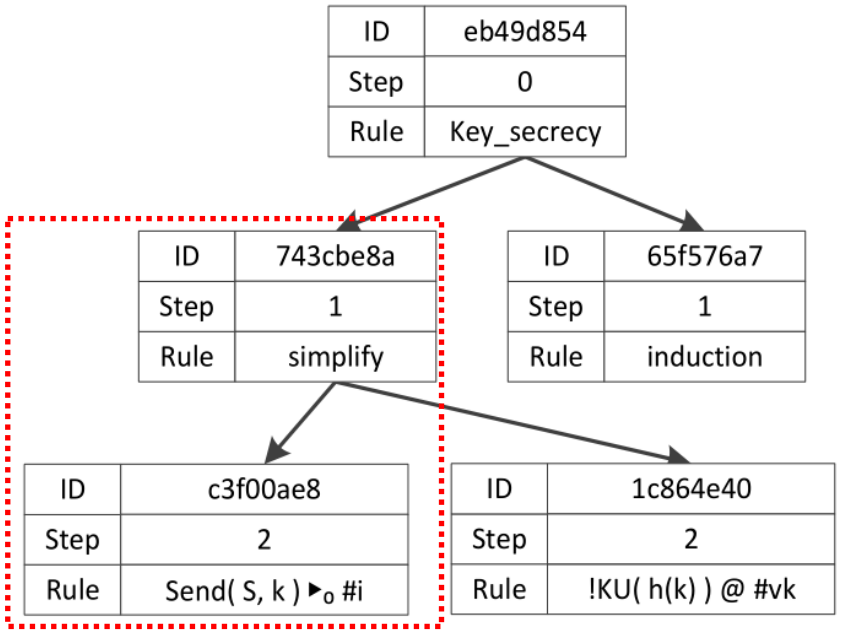
- The root node represents the security property



Problem Definition

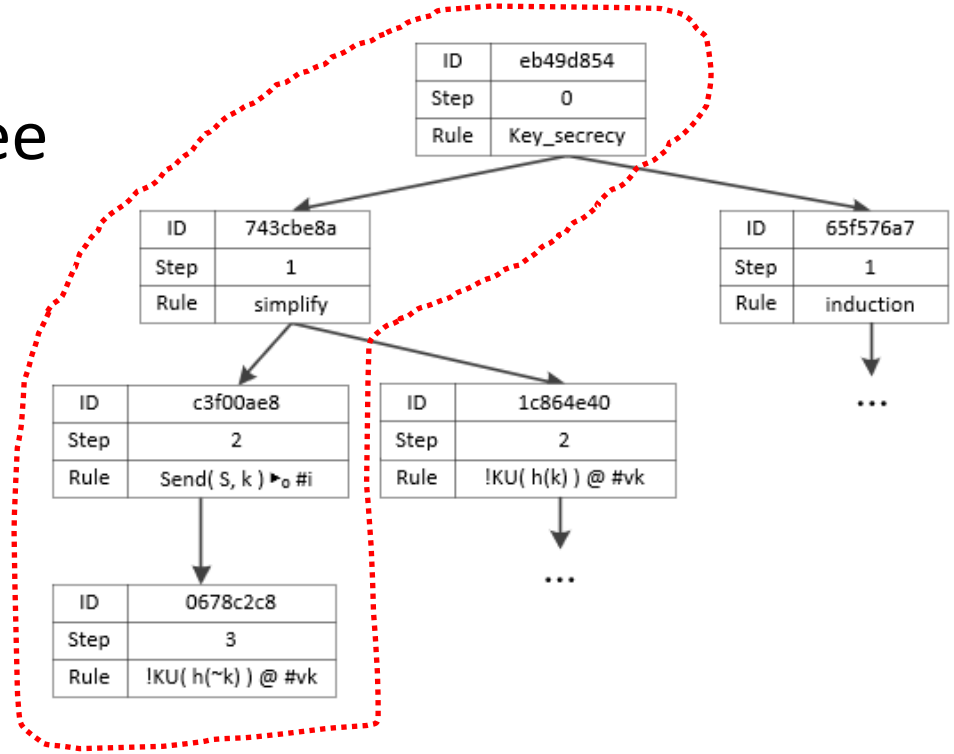
- Verification process in our work
  - The proof state in the child can be used to prove the proof state in its parent
  - E.g.

Send(S, k) $\blacktriangleright_0$ #i ⇨ simplify



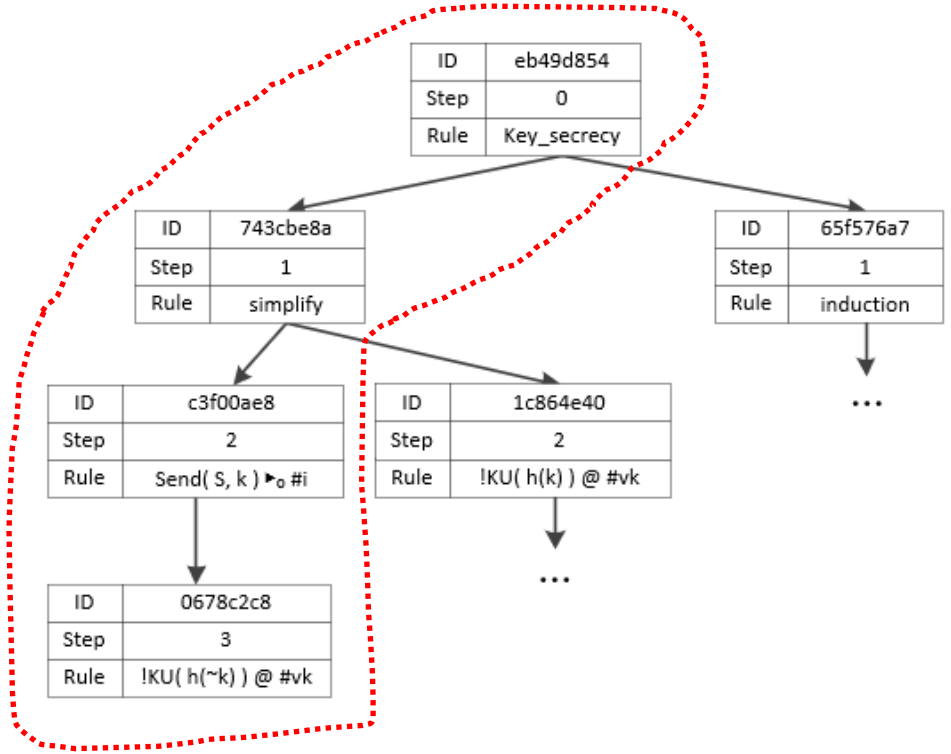
Problem Definition

- Goal: finding a correct proof path in the tree
  - Correct path
    - The path starts from the root node and ends with a leaf node
    - The leaf node contains an axiom



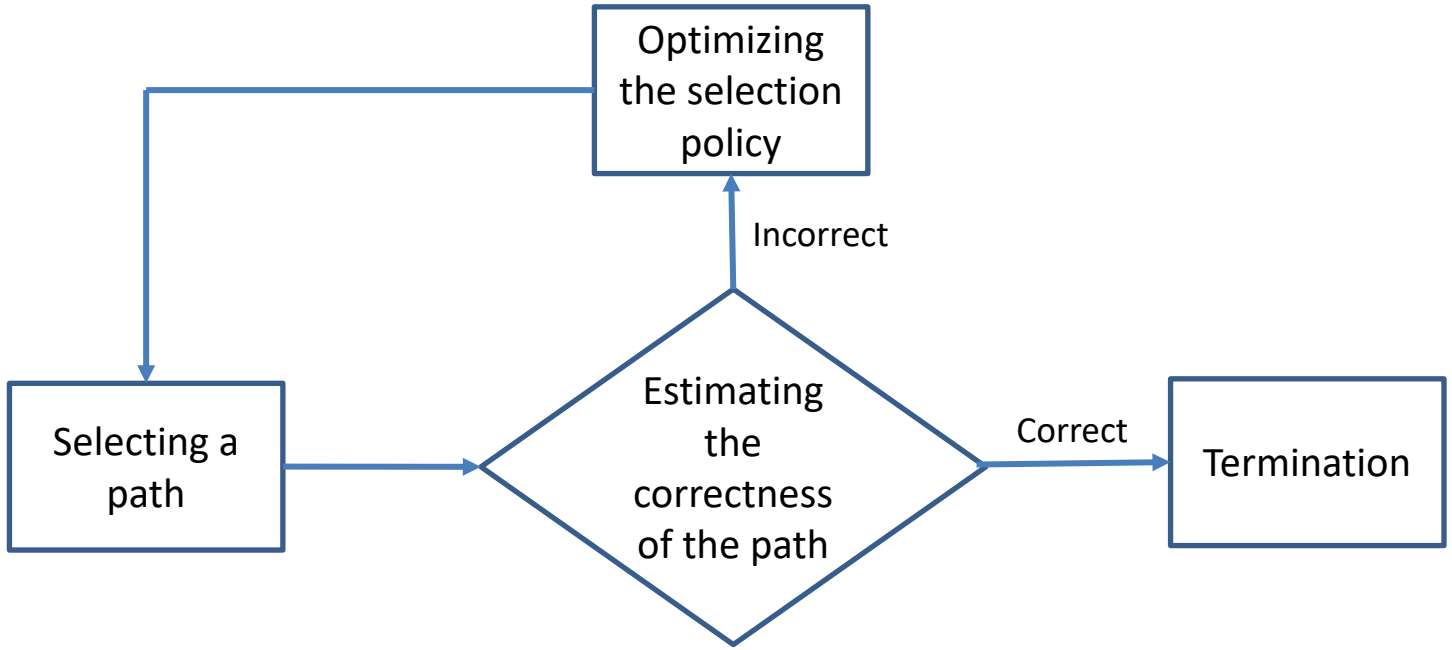
Problem Definition

- Supporting lemma
  - If a node is on a correct path, the proposition it contains is a supporting lemma



Strategy Design

- Our dynamic strategy
  - Finding the correct proof path by optimizing itself according to historical incorrect paths

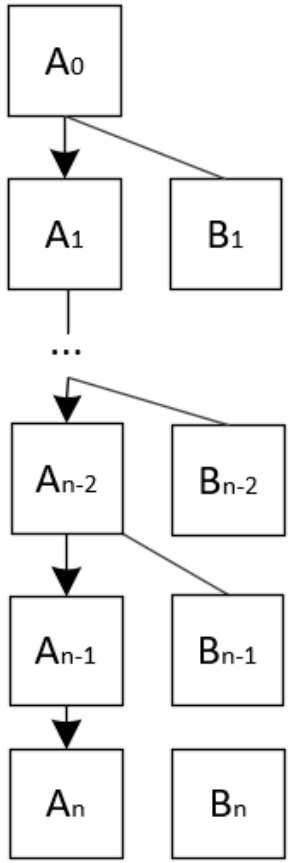


Insight

- Formal
  - Proven in Appendix.A

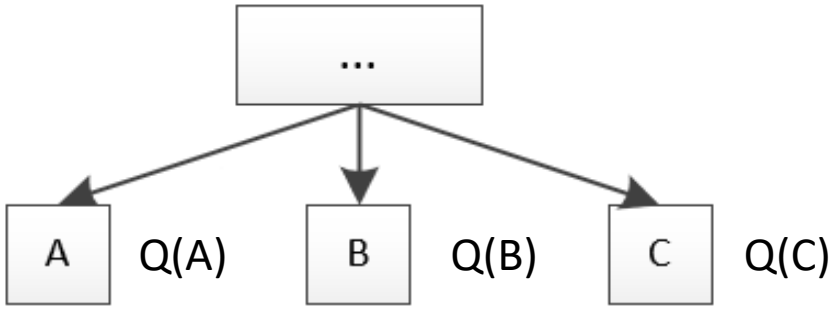
    The node representing a supporting lemma is on the incorrect path with lower probability, when a random strategy is given.

- Informal
  - When $[A_0,A_1,\ldots,A_{n-1},A_n]$ is estimated as an incorrect path
  - The probabilities of $A_1,\ldots,A_{n-1}$ being supporting lemmata decrease exponentially



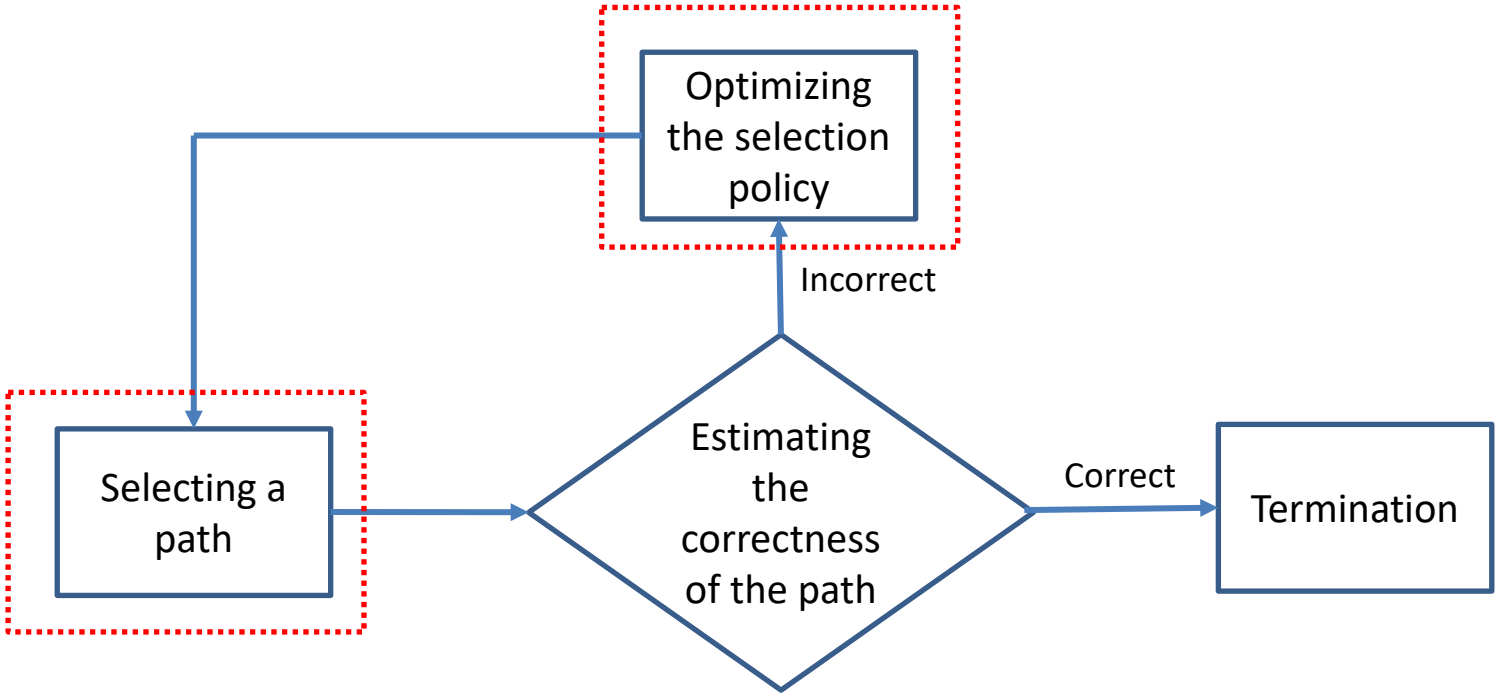
Our Method

- We apply our insight by introducing Deep Q Network (DQN)
  - DQN: a reinforcement learning agent
    - Computes a Q value for each node
    - Tends to select nodes with higher Q values
    - Adjusts Q values by optimizing itself



Our Method

- We use the DQN to select the path and optimize the DQN in our dynamic strategy

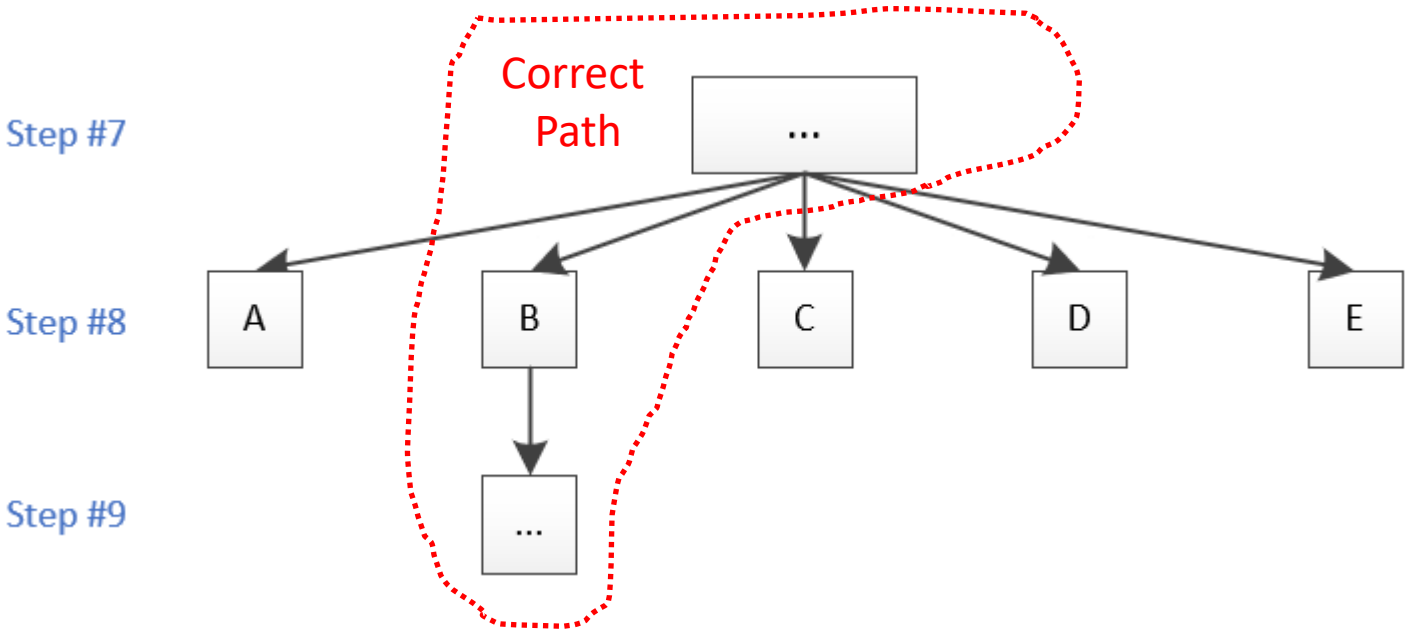


An Example

- Yubikey Protocol
  - The most important case in our experiment
  - Widely studied
    - Künnemann et al., LNCS, 2013
    - Kremer et al., IEEE S&P, 2014
    - Bruni et al., IEEE CSF, 2015
    - Cheval et al., IEEE CSF, 2018
  - Still cannot be automatically verified by all the state-of-the-art tools

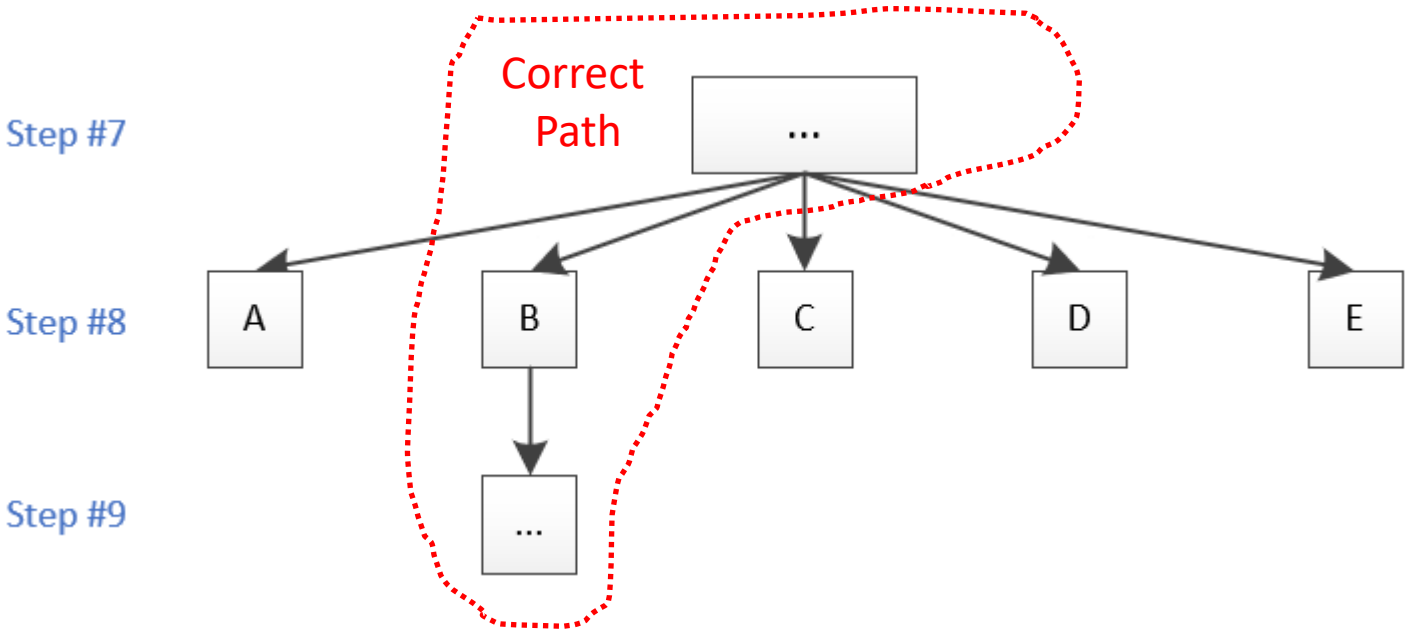
An Example

- Part of the verification tree
  - A, C, D, E represent unsupporting lemmata



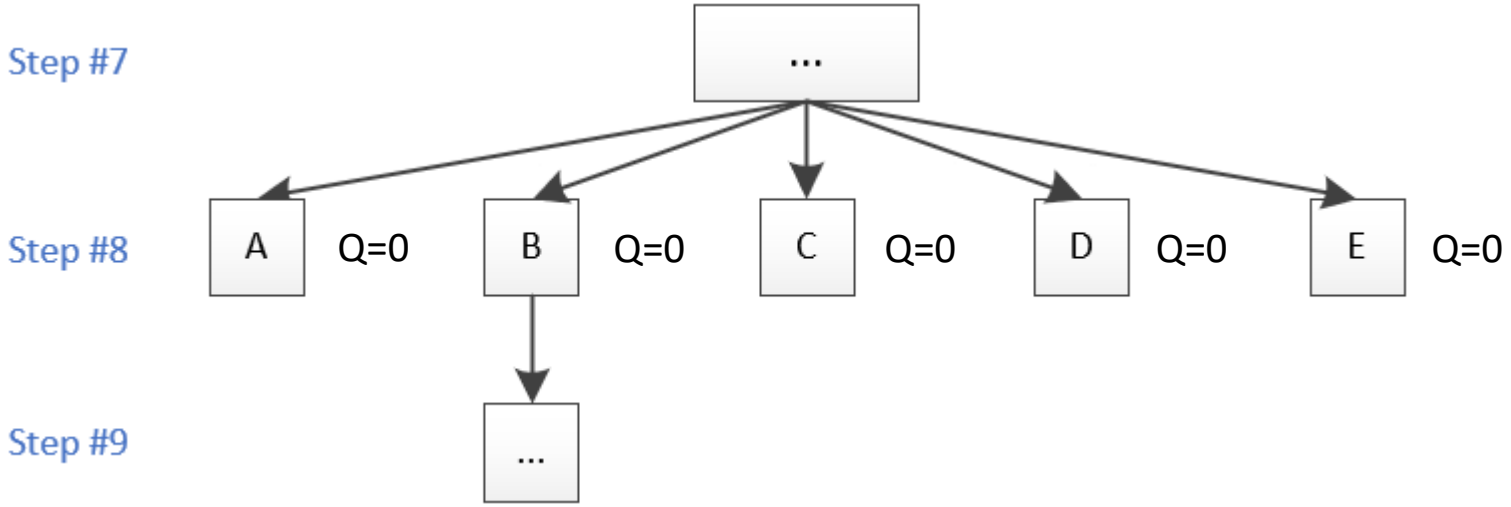
An Example

- Tamarin Prover
  - Selected A in Step #8
    - Lead to a loop in verification



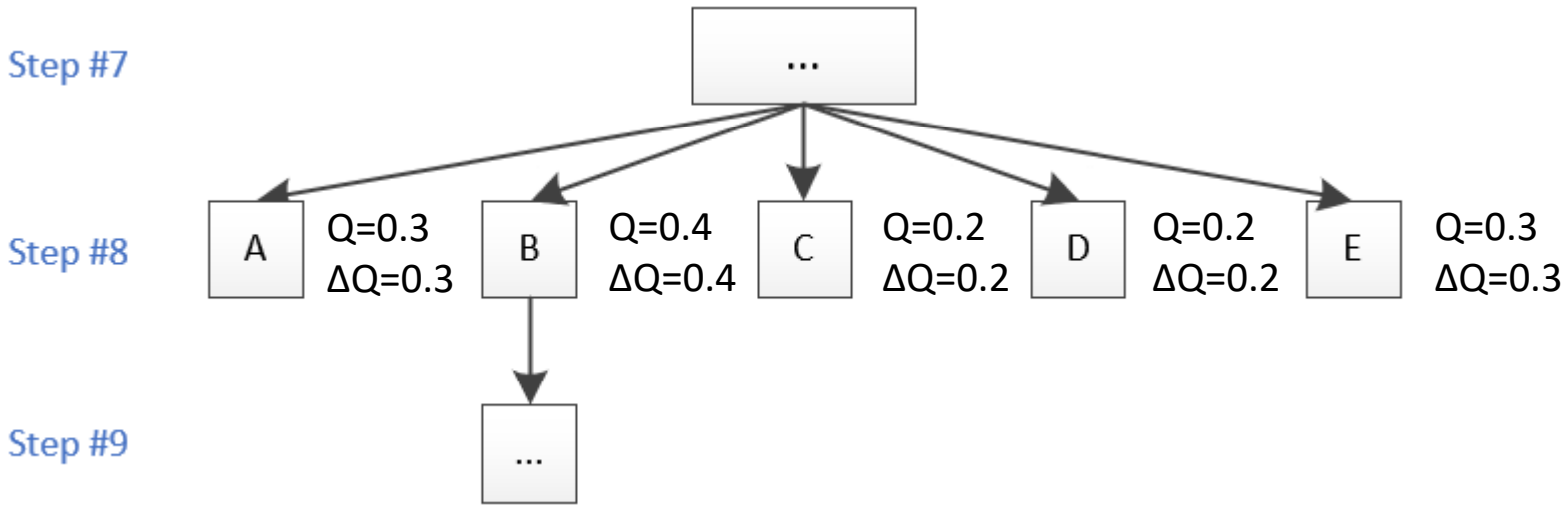
An Example

- SmartVerif
  - Initial epoch



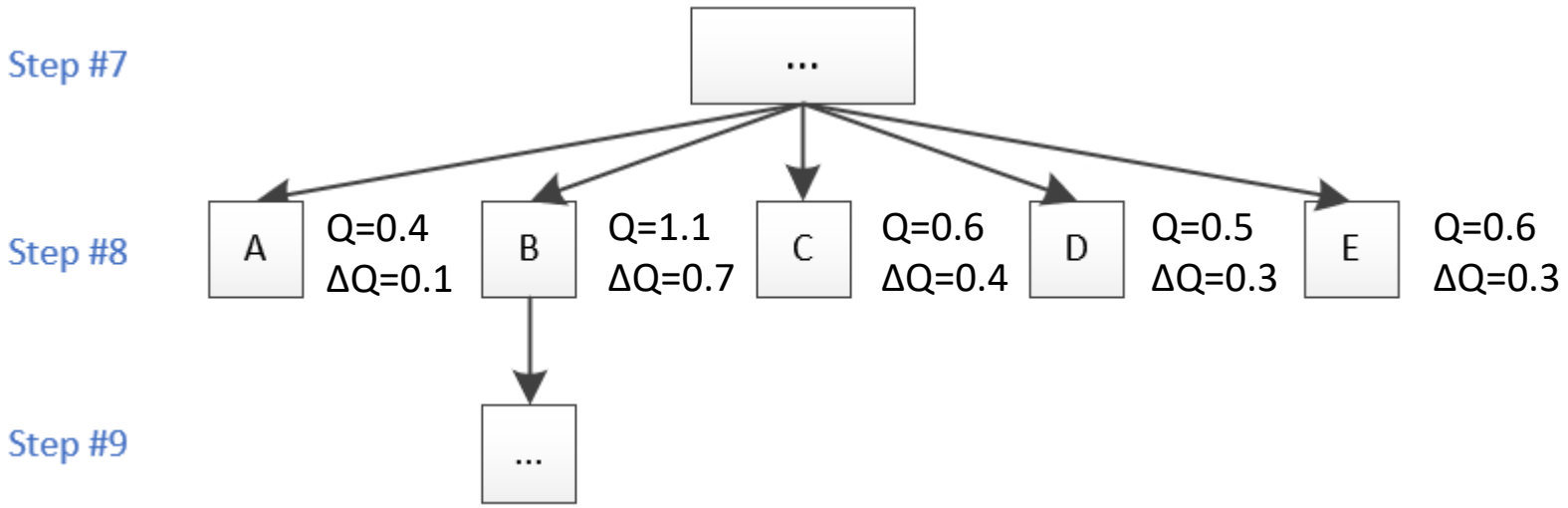
An Example

- Epoch 20



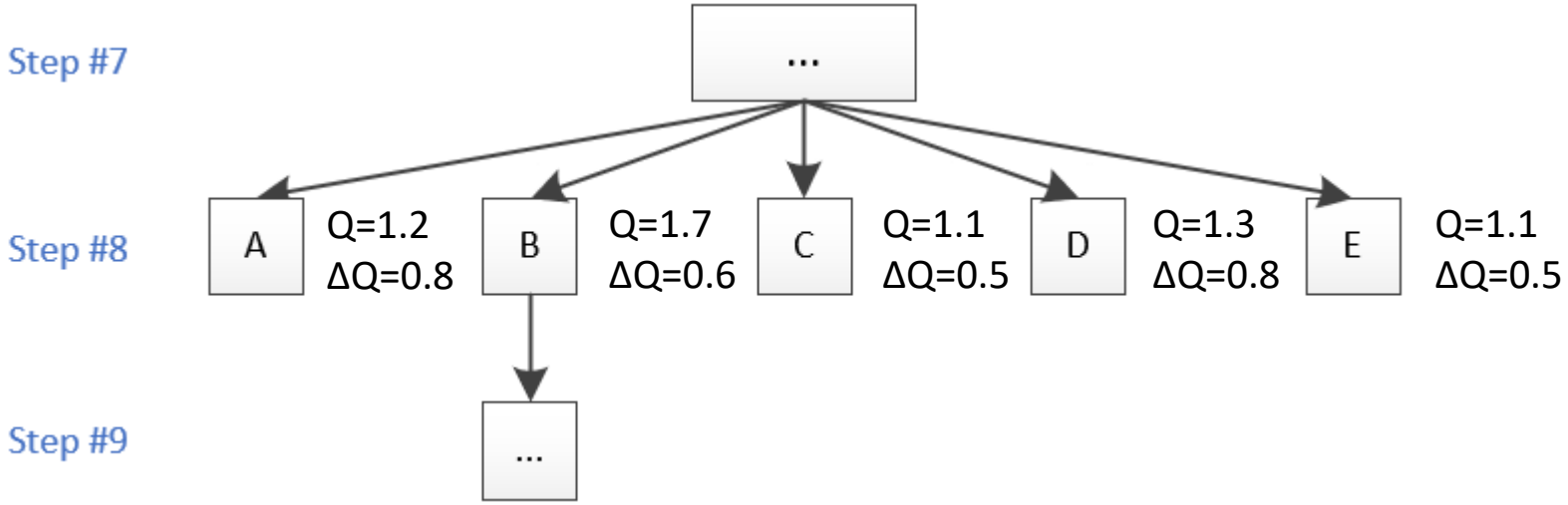
An Example

- Epoch 40



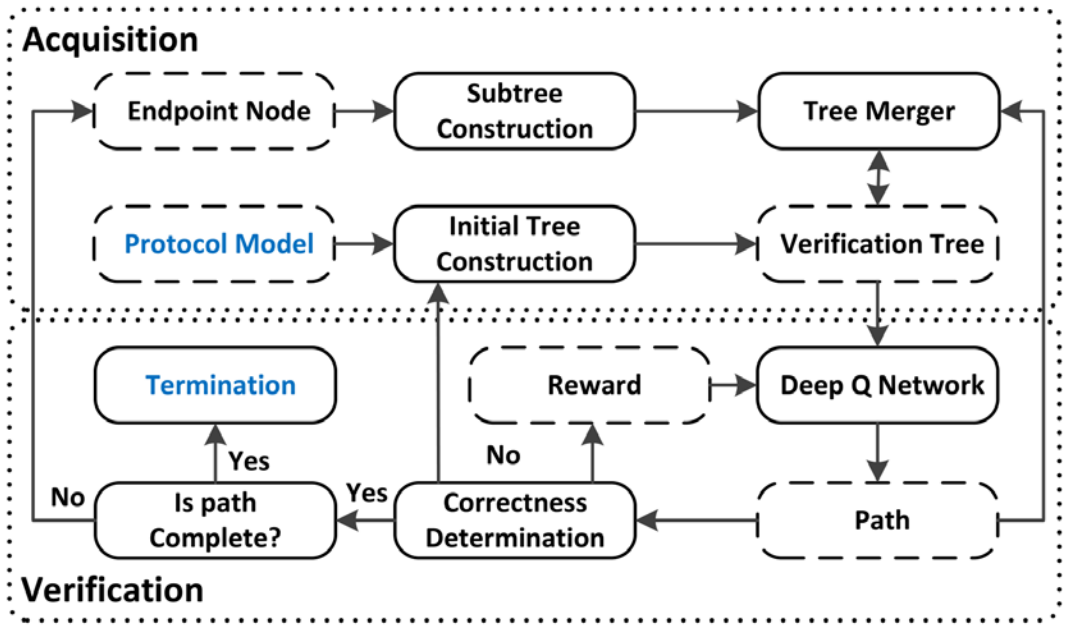
An Example

- Epoch 79
  - Finds the correct proof path



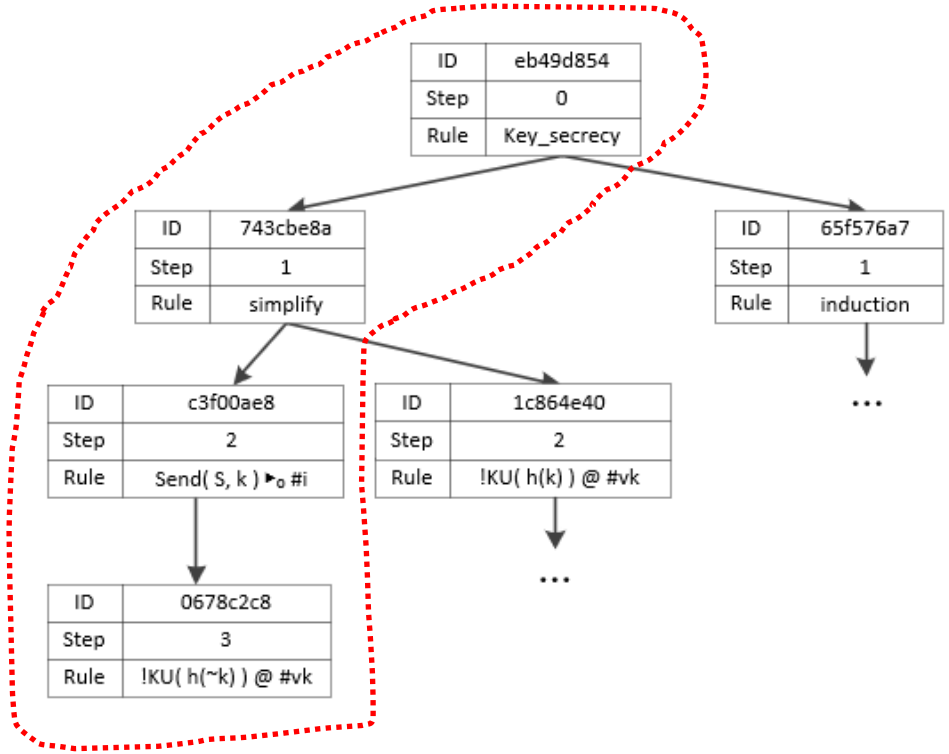
Implementation

- We solve several technical problems
  - Constructing verification tree
  - Estimating correctness of the path
  - Designing rewards for DQN



Implementation

- Constructing the verification tree
  - The tree is generated and expanded gradually
    - In each round, only one of the endpoint nodes in the current tree is expanded
    - The selection of the endpoint node is guided by the DQN



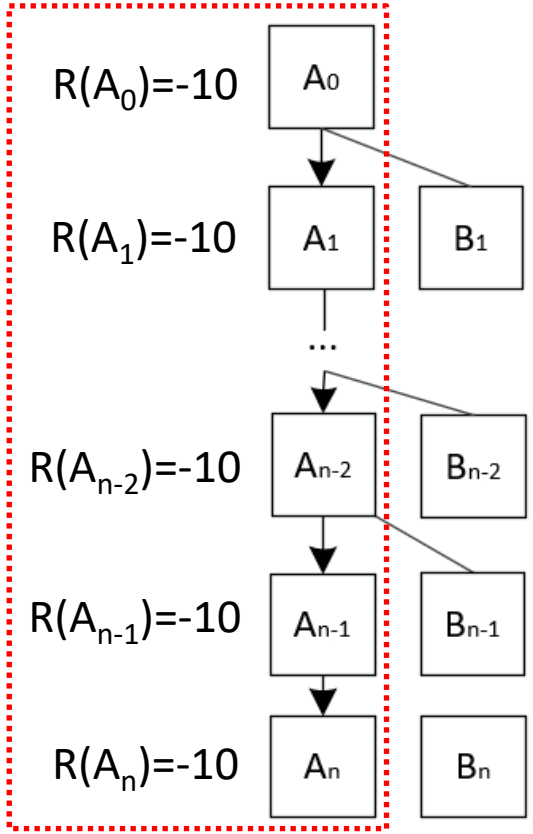
Implementation

- Estimating the correctness of the path
  - Loop Detection Algorithm
    - Counting the number of similar elements on the path

```
case I_2
solve( !KU( aenc(<'2', ~ni.1, nr.1, $R.1>,
                 pk(~ltkA.1))
        ) @ #vk.1 )
  case I_2
  solve( !KU( aenc(<'2', ~ni.2, nr.2, $R.2>,
                 pk(~ltkA.4))
        ) @ #vk.3 )
    case I_2
    solve( !KU( aenc(<'2', ~ni.3, nr.3, $R.3>,
                 pk(~ltkA.7))
        ) @ #vk.5 )
      case I_2
      solve( !KU( aenc(<'2', ~ni.4, nr.4, $R.4>,
                 pk(~ltkA.10))
        ) @ #vk.7 )
        case I_2
        solve( !KU( aenc(<'2', ~ni.5, nr.5, $R.5>,
                 pk(~ltkA.13))
        ) @ #vk.9 )
```

Implementation

- Designing rewards for DQN
  - When $[A_0, A_1, \ldots, A_{n-1}, A_n]$ is estimated as an incorrect path
  - Gives a negative reward for all the nodes on the path



Experiments

Chosen tools

- StatVerif, Set-pi and GSVerif
- Tamarin Prover with different heuristics(‘s’, ‘c’, ‘p’)
- SmartVerif

Chosen protocols

- All the protocols that have been evaluated in papers of the compared tools
- Five protocols with observation equivalence properties

Experiments

- The success rate of automatically verifying 24 protocols with unbounded sessions

	Stat Verif	Set-pi	GSVerif	Tamarin ('s')	Tamarin ('c')	Tamarin ('p')	Smart Verif
Success Rate	3/24	5/24	16/24	14/24	14/24	16/24	24/24

- SmartVerif can fully automatically verify all the protocols

Experiments

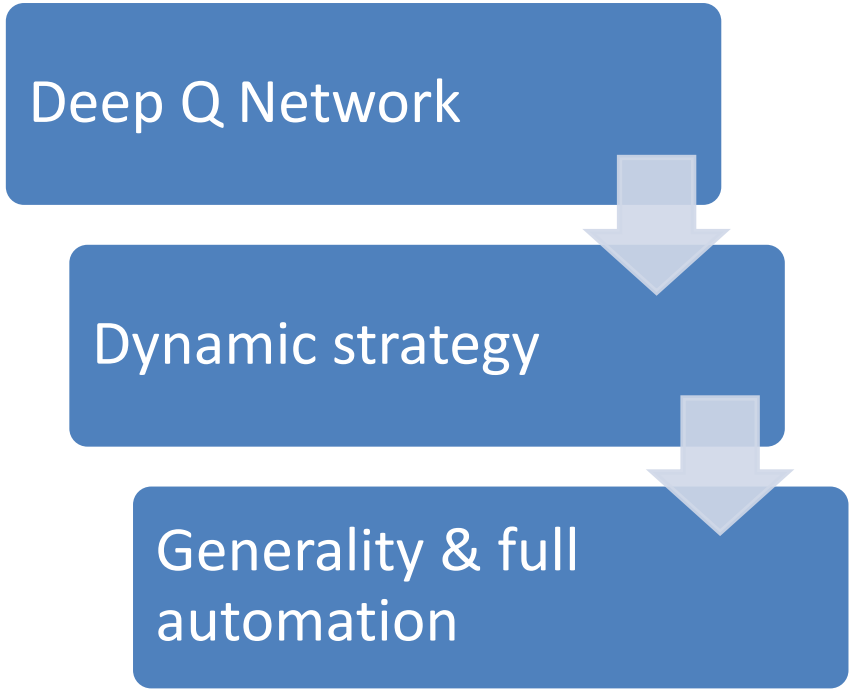
- Except 24 protocol cases, there are many practical protocols
  - TLS 1.3
  - 5G AKA
  - Smart contract
  - Blockchain protocol
- Cannot be automatically verified by state-of-the-art tools
- SmartVerif can fully automatically verify these protocols

Experiments

- Efficiency and overhead of SmartVerif
  - Two metrics
    - Running time
    - Training epochs
  - SmartVerif verifies protocols in a very efficient way
    - For most protocols, epochs < 25, time < 0.5 hour
    - The worst case (PKCS #11): 175 epochs, 83 minutes

Conclusion

- Problem: automatically verify security protocols
- Method: a dynamic strategy
- Contributions
  - We present, to the best of our knowledge, the first framework that automatically verifies security protocols by dynamic strategies
  - We achieve our dynamic strategy by using the DQN and designing the rewards and the algorithm that estimate the correctness of paths
  - Our work achieve generality in designing heuristics and full automation in verification



Future Work

- Using an optimized static strategy as the initial strategy in SmartVerif
- Applying our dynamic strategy to other problems, e.g. automated formal verification of software or systems
- Optimizing the efficiency of SmartVerif

Q & A

Thanks!

Reporter: Cheng Su

gotzeus@mail.ustc.edu.cn