Centrum Wiskunde & Informatica

Erasable PUFs: Formal Treatment and Generic Design

Chenglu Jin, Wayne Burleson, Marten van Dijk, and Ulrich Rührmair

Physical Unclonable Functions (PUFs)

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations
- Ideally, PUFs are Physical Random Functions

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations
- Ideally, PUFs are Physical Random Functions

Applications

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations
- Ideally, PUFs are Physical Random Functions

Applications

- Device/Chip Authentication

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations
- Ideally, PUFs are Physical Random Functions

Applications

- Device/Chip Authentication
- Key Management/Storage

Physical Unclonable Functions (PUFs)

- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations
- Ideally, PUFs are Physical Random Functions

Applications

- Device/Chip Authentication
- Key Management/Storage
- Cryptographic Protocols (Key Exchange, Oblivious Transfer, Bit Commitment)

Physical Unclonable Functions (PUFs)

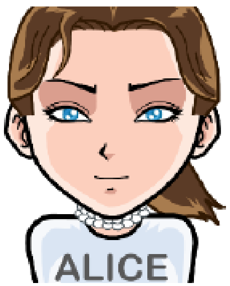
- Hardware security primitive taking challenges and generating responses
- Unique fingerprint on individual IC even if designed and fabricated in the same way
- Leveraging manufacturing process variations
- Ideally, PUFs are Physical Random Functions

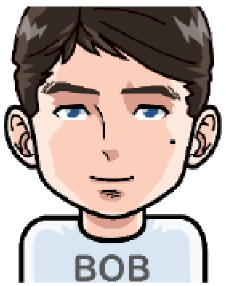
Applications

- Device/Chip Authentication
- Key Management/Storage
- Cryptographic Protocols (Key Exchange, Oblivious Transfer, Bit Commitment)

Today's Focus

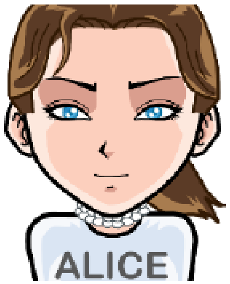
Simplified PUF-based Key Exchange Protocol

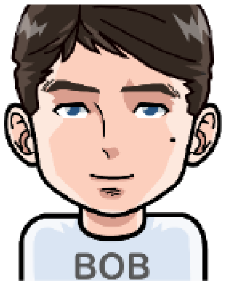




$C_0, C_1, \ldots, C_k$ →  → $R_0, R_1, \ldots, R_k$

Simplified PUF-based Key Exchange Protocol

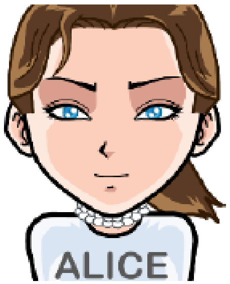


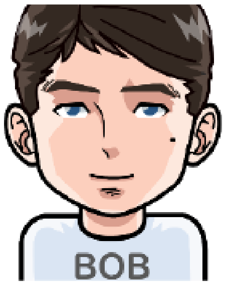


(C, R)

Public, Authenticated Physical Channel

Simplified PUF-based Key Exchange Protocol



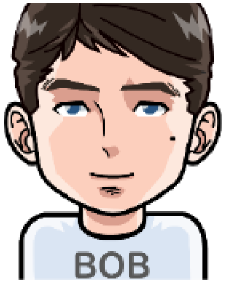


(C, R)

$C_0, C_1, \ldots, C_k$

Public, Authenticated Communication Channel

Simplified PUF-based Key Exchange Protocol



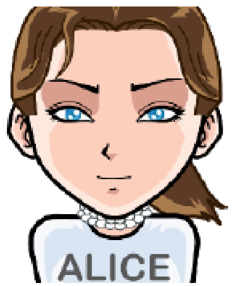
(C, R)

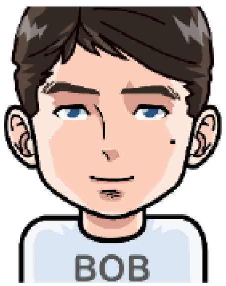
$C_0, C_1, \ldots, C_k$ →  → $R_0, R_1, \ldots, R_k$

R is the shared secret key

The security of this protocol relies on the unpredictability of PUF responses given its challenges.

Simplified PUF-based Key Exchange Protocol





(C, R)

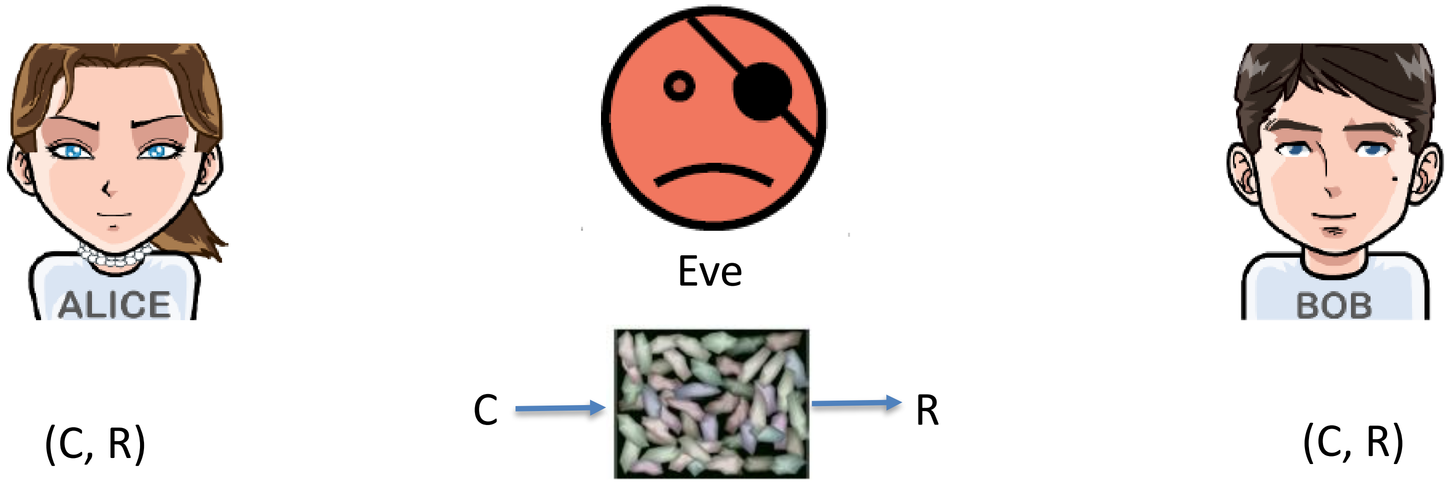
$C_0, C_1, \ldots, C_k$ →  → $R_0, R_1, \ldots, R_k$

R is the shared secret key

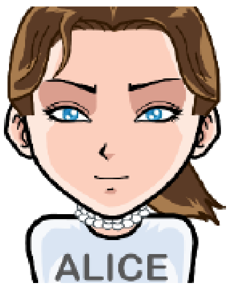
The security of this protocol relies on the unpredictability of PUF responses given its challenges.

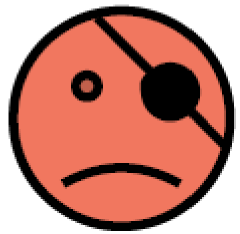
Not Complete!

After Protocol Execution

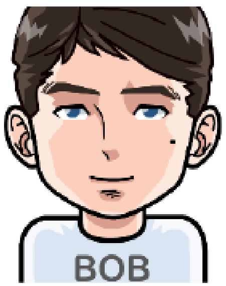


After Protocol Execution





Eve



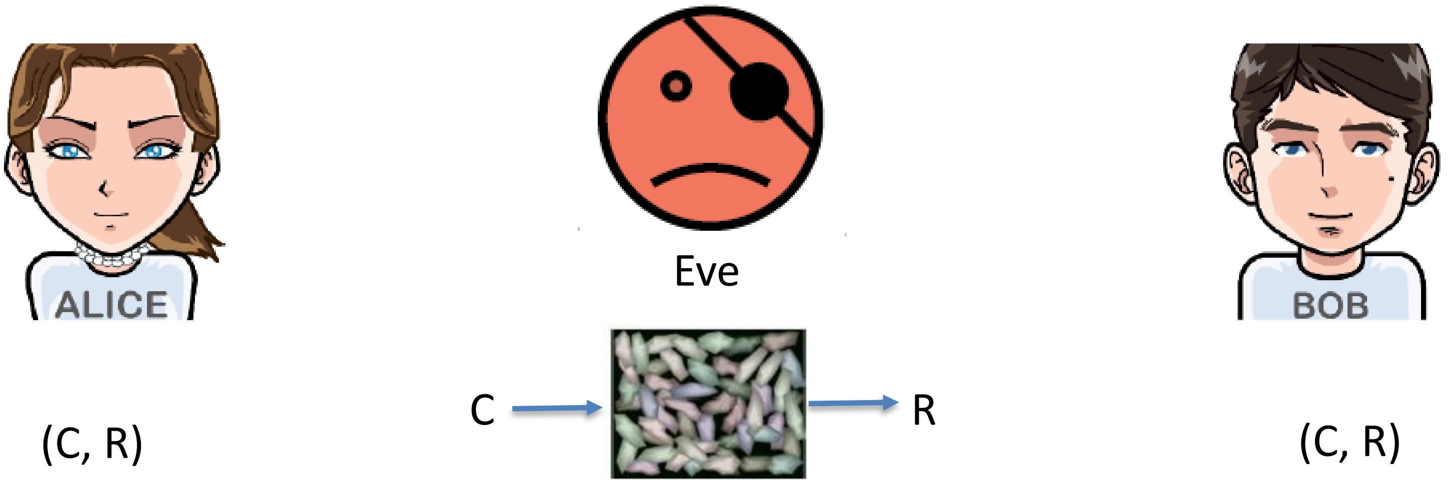
(C, R)

C  R

(C, R)

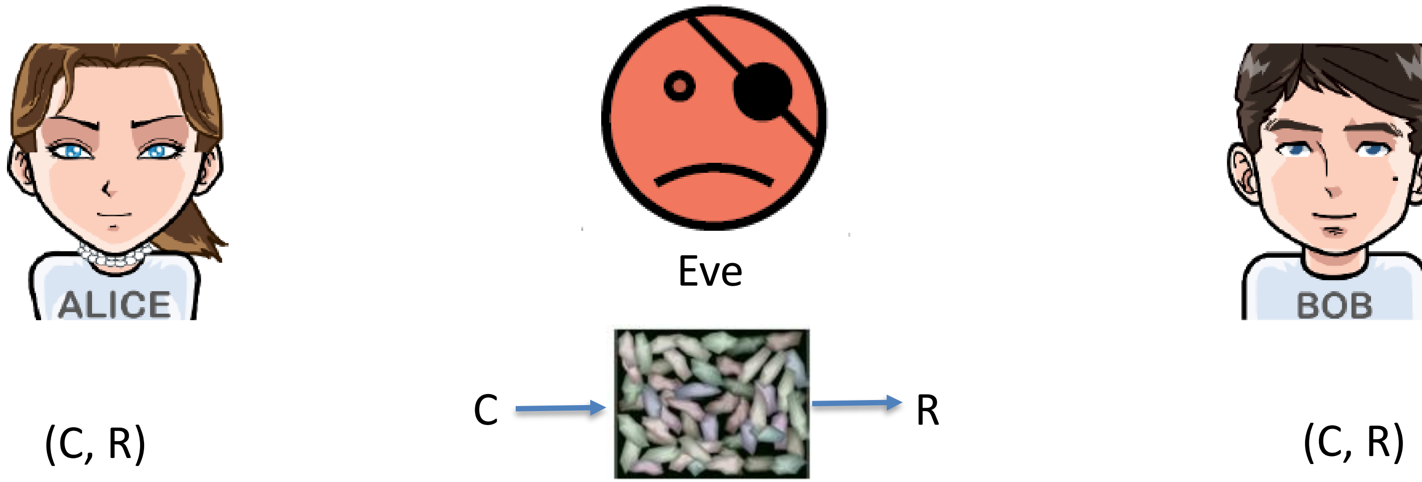
- In the PUF Re-Use model, Eve can know the secret R as well.

After Protocol Execution



- In the PUF Re-Use model, Eve can know the secret R as well.
- Highly realistic threat against PUF-based protocol, as destroying PUFs after one protocol execution is prohibitively uneconomic.

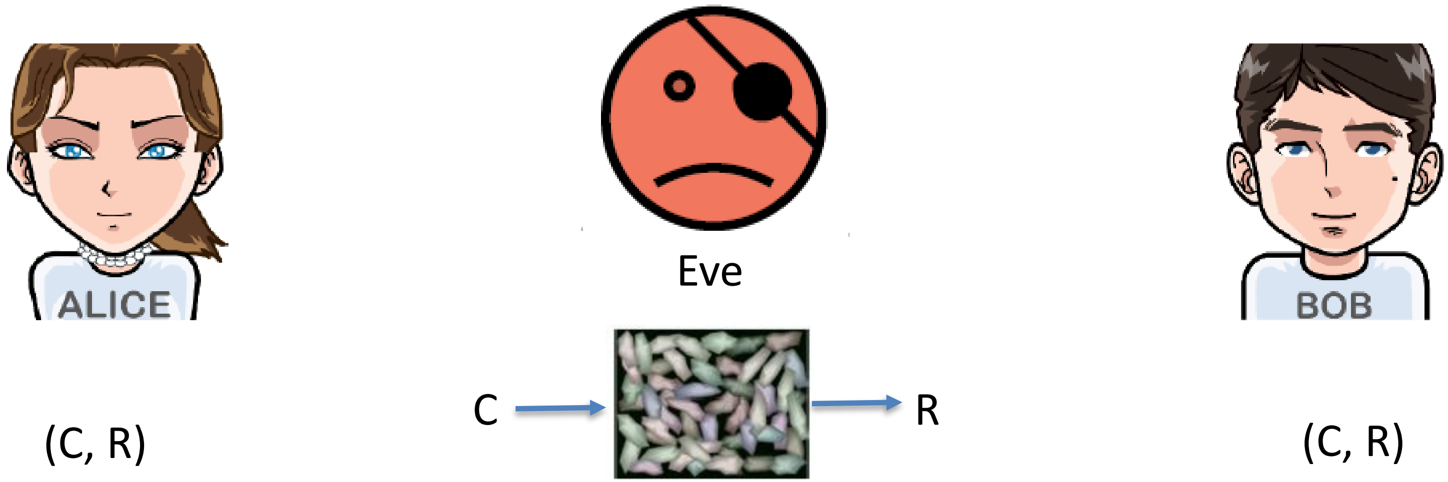
After Protocol Execution



- In the PUF Re-Use model, Eve can know the secret R as well.
- Highly realistic threat against PUF-based protocol, as destroying PUFs after one protocol execution is prohibitively uneconomic.
- Actually, impossibility results of constructing PUF-based crypto protocols like KE/OT in PUF Re-Use model have been proved.

Marten van Dijk and Ulrich Rührmair. "Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results." IACR ePrint (2012)

After Protocol Execution



- In the PUF Re-Use model, Eve can know the secret R as well.
- Highly realistic threat against PUF-based protocol, as destroying PUFs after one protocol execution is prohibitively uneconomic.
- Actually, impossibility results of constructing PUF-based crypto protocols like KE/OT in PUF Re-Use model have been proved.
- The issue has to be solved on the hardware level.

Marten van Dijk and Ulrich Rührmair. "Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results." IACR ePrint (2012)

Basic Idea of Effective Countermeasures

Basic Idea of Effective Countermeasures

- Erase the CRP used in the protocol execution after the protocol

Basic Idea of Effective Countermeasures

- Erase the CRP used in the protocol execution after the protocol
- Attackers will have no way to re-access the secret response value

Basic Idea of Effective Countermeasures

- Erase the CRP used in the protocol execution after the protocol
- Attackers will have no way to re-access the secret response value

- Can a reconfigurable PUF solve the problem?

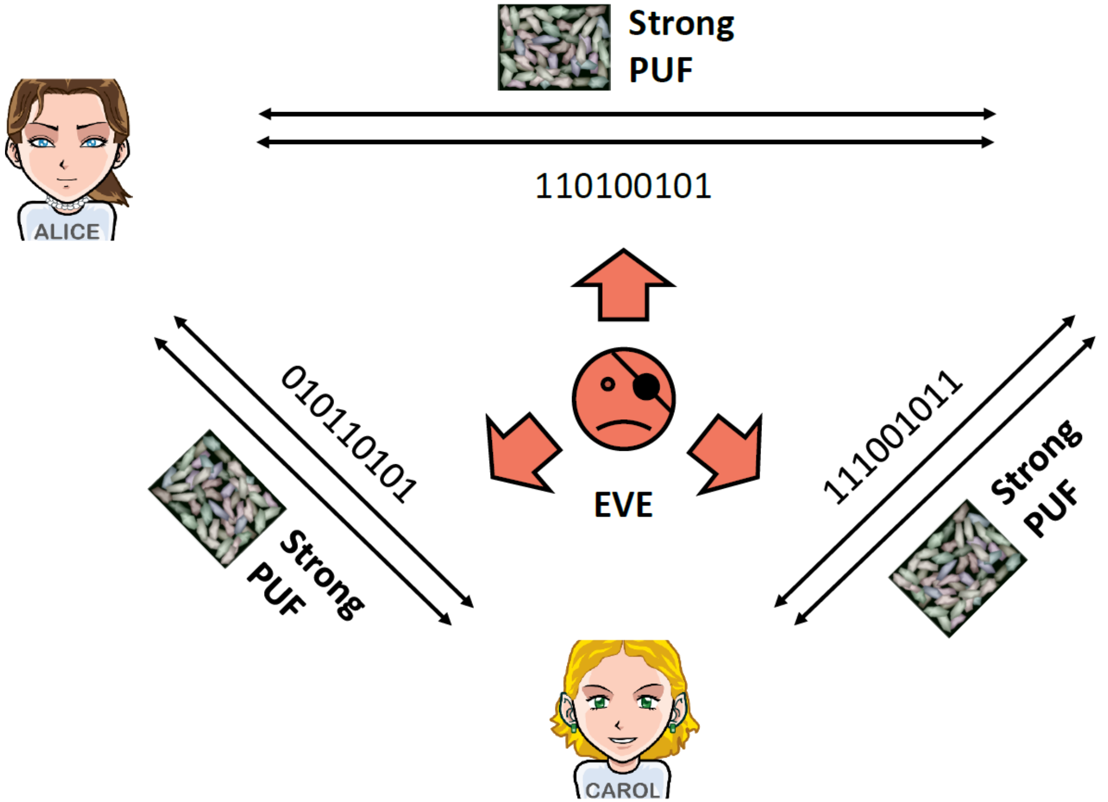
Basic Idea of Effective Countermeasures

- Erase the CRP used in the protocol execution after the protocol
- Attackers will have no way to re-access the secret response value

- Can a reconfigurable PUF solve the problem?
- A Reconfigurable PUF allows users to alter the responses of all challenges in one single operation (so-called “Reconfiguration”).

Stefan Katzenbeisser, *et al.* "Recyclable pufs: Logically reconfigurable pufs." Journal of Cryptographic Engineering (2011)

Multi-Party Use Case



Using reconfigurable PUFs in crypto protocols cannot support multi-party use case.

Erasable PUFs

- Allows users to erase/alter the response of individual challenges chosen by the users

Erasable PUFs

- Allows users to erase/alter the response of individual challenges chosen by the users
- Erasable PUF-based crypto protocols can allow multiple parties to share one PUF and avoid repeated physical transfer of the PUF

Erasable PUFs

- Allows users to erase/alter the response of individual challenges chosen by the users
- Erasable PUF-based crypto protocols can allow multiple parties to share one PUF and avoid repeated physical transfer of the PUF
- Users can only erase the used CRPs after protocol execution, without affecting the other CRPs

Basic Idea to Realize Erasable PUFs

Basic Idea to Realize Erasable PUFs

- Add an interface around a PUF to enforce access control to the PUF

Basic Idea to Realize Erasable PUFs

- Add an interface around a PUF to enforce access control to the PUF
- Create a list of erased challenges

Basic Idea to Realize Erasable PUFs

- Add an interface around a PUF to enforce access control to the PUF
- Create a list of erased challenges
- If a queried challenge is in the list of erased challenges, then the interface should deny the access to the PUF

Basic Idea to Realize Erasable PUFs

- Add an interface around a PUF to enforce access control to the PUF
- Create a list of erased challenges
- If a queried challenge is in the list of erased challenges, then the interface should deny the access to the PUF
- Otherwise, the interface will allow the PUF to be queried, and the response will be generated and outputted.

Basic Idea to Realize Erasable PUFs

- Add an interface around a PUF to enforce access control to the PUF
- Create a list of erased challenges
- If a queried challenge is in the list of erased challenges, then the interface should deny the access to the PUF
- Otherwise, the interface will allow the PUF to be queried, and the response will be generated and outputted.
- Add new challenges into the list to erase them logically

Basic Idea to Realize Erasable PUFs

- Add an interface around a PUF to enforce access control to the PUF
- Create a list of erased challenges
- If a queried challenge is in the list of erased challenges, then the interface should deny the access to the PUF
- Otherwise, the interface will allow the PUF to be queried, and the response will be generated and outputted.
- Add new challenges into the list to erase them logically

- Drawback: The list should not be tampered with by adversaries, but the size of the list is growing when more and more challenges are erased. This implies that a large trusted memory is needed

Our Solution: Genie PUFs

Our Solution: Genie PUFs

- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs

Our Solution: Genie PUFs

- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs
- Goal: Reduce the size of trusted memory in the trusted computing base (TCB)

Our Solution: Genie PUFs

- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs
- Goal: Reduce the size of trusted memory in the trusted computing base (TCB)
- Key Idea: Merge Authenticated Search Tree and Red-Black Tree structure to securely outsource the list of erased challenges to untrusted memory

Our Solution: Genie PUFs

- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs
- Goal: Reduce the size of trusted memory in the trusted computing base (TCB)
- Key Idea: Merge Authenticated Search Tree and Red-Black Tree structure to securely outsource the list of erased challenges to untrusted memory

- What can we achieve?

Our Solution: Genie PUFs

- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs
- Goal: Reduce the size of trusted memory in the trusted computing base (TCB)
- Key Idea: Merge Authenticated Search Tree and Red-Black Tree structure to securely outsource the list of erased challenges to untrusted memory

- What can we achieve?
- Only require a constant-sized trusted memory in the TCB to store the root hash of the tree structure

Our Solution: Genie PUFs

- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs
- Goal: Reduce the size of trusted memory in the trusted computing base (TCB)
- Key Idea: Merge Authenticated Search Tree and Red-Black Tree structure to securely outsource the list of erased challenges to untrusted memory

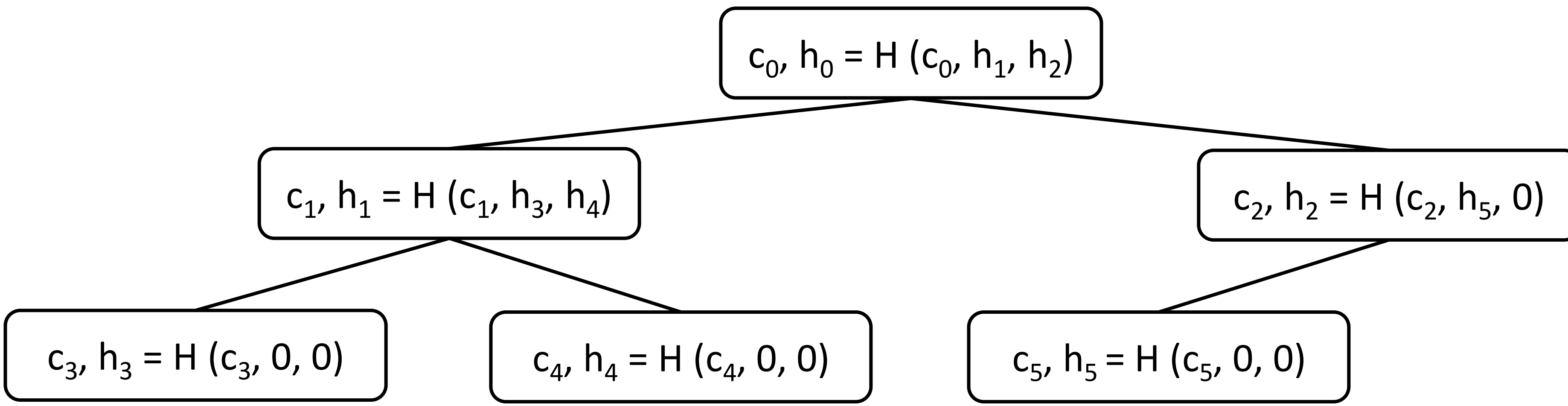
- What can we achieve?
- Only require a constant-sized trusted memory in the TCB to store the root hash of the tree structure
- Support arbitrarily large list of erased challenges

Our Solution: Genie PUFs

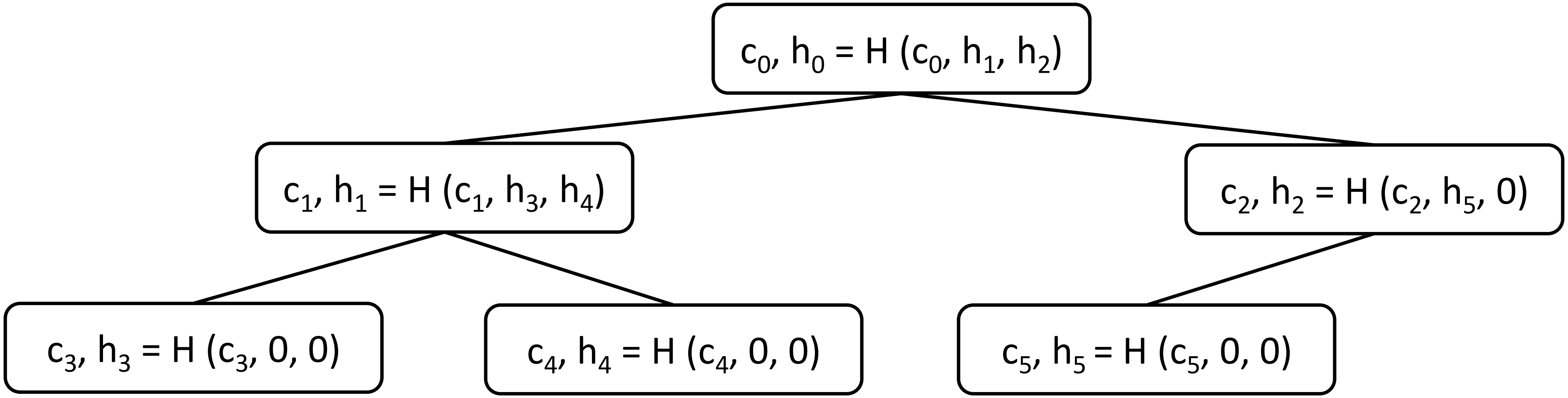
- Generic Erasable PUFs (Genie PUFs), because its just a PUF interface, and it can be integrated with any PUFs
- Goal: Reduce the size of trusted memory in the trusted computing base (TCB)
- Key Idea: Merge Authenticated Search Tree and Red-Black Tree structure to securely outsource the list of erased challenges to untrusted memory

- What can we achieve?
- Only require a constant-sized trusted memory in the TCB to store the root hash of the tree structure
- Support arbitrarily large list of erased challenges
- Using the combined tree structure, the untrusted memory can provide a O(log(N)) size proof to the TCB to prove a challenge is (not) in the list of size N

Authenticated Search Tree Construction

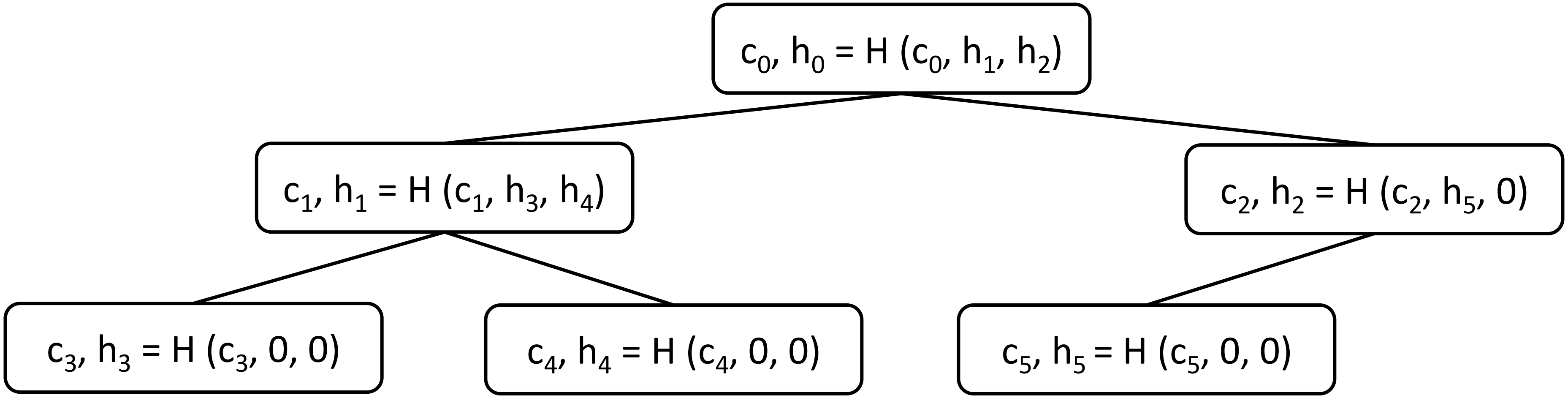


Authenticated Search Tree Construction



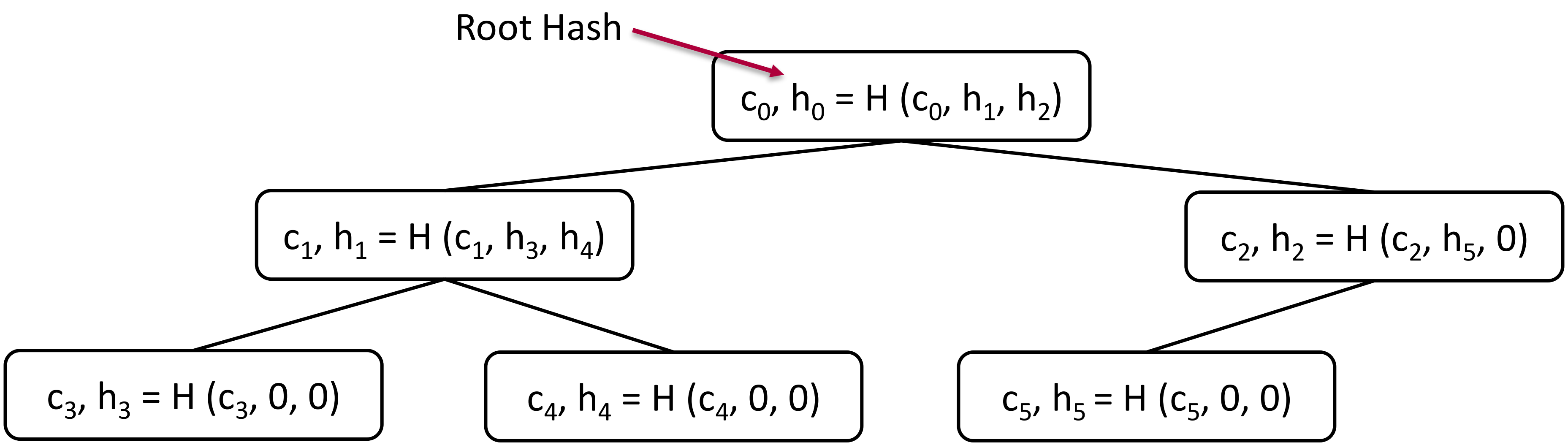
- In each node of the tree, we store one unique challenge, and the tree is sorted like a binary search tree according to the challenge value in each node

Authenticated Search Tree Construction



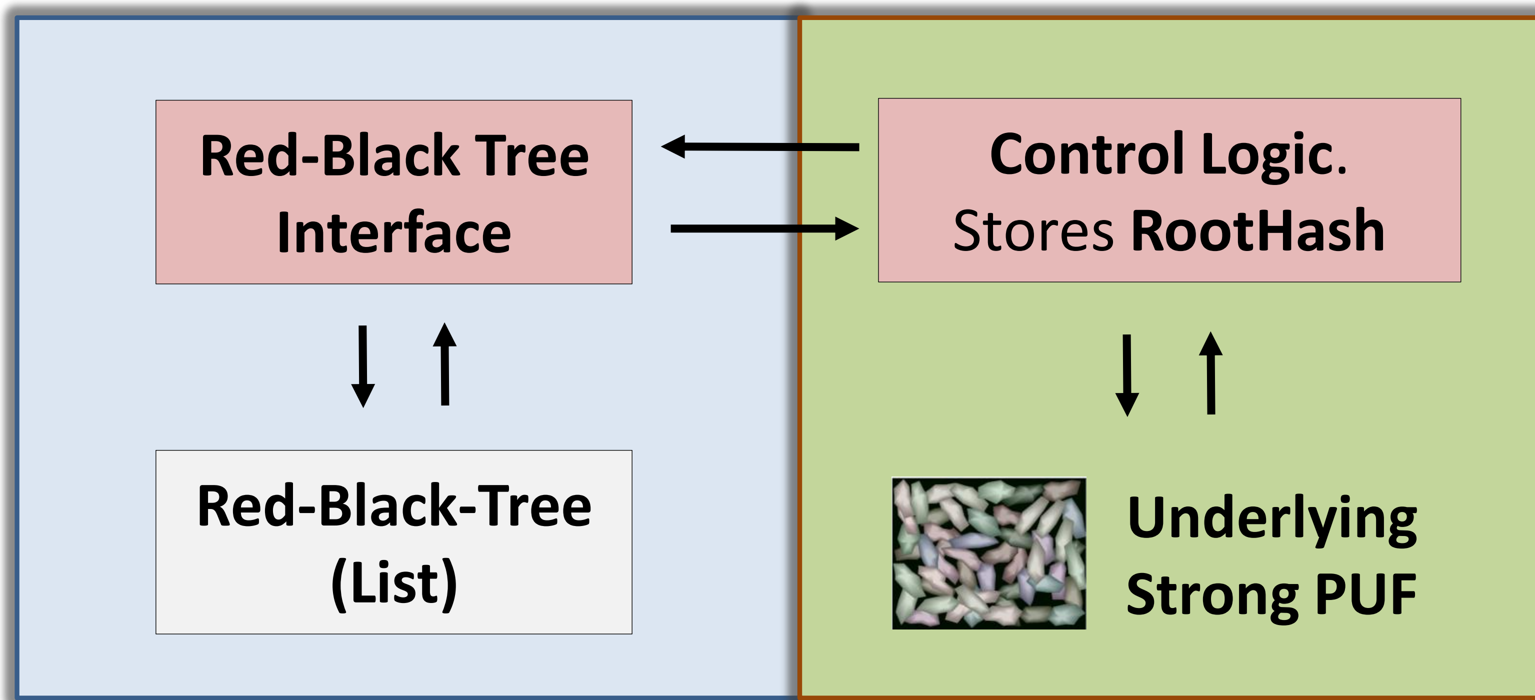
- In each node of the tree, we store one unique challenge, and the tree is sorted like a binary search tree according to the challenge value in each node
- Besides the challenge c_i, a hash value is stored in each node, where $h_i = H(c_i$, hash value stored in its left child, hash value stored in its right child)

Authenticated Search Tree Construction



- In each node of the tree, we store one unique challenge, and the tree is sorted like a binary search tree according to the challenge value in each node
- Besides the challenge c_i , a hash value is stored in each node, where $h_i = H$ (c_i, hash value stored in its left child, hash value stored in its right child)

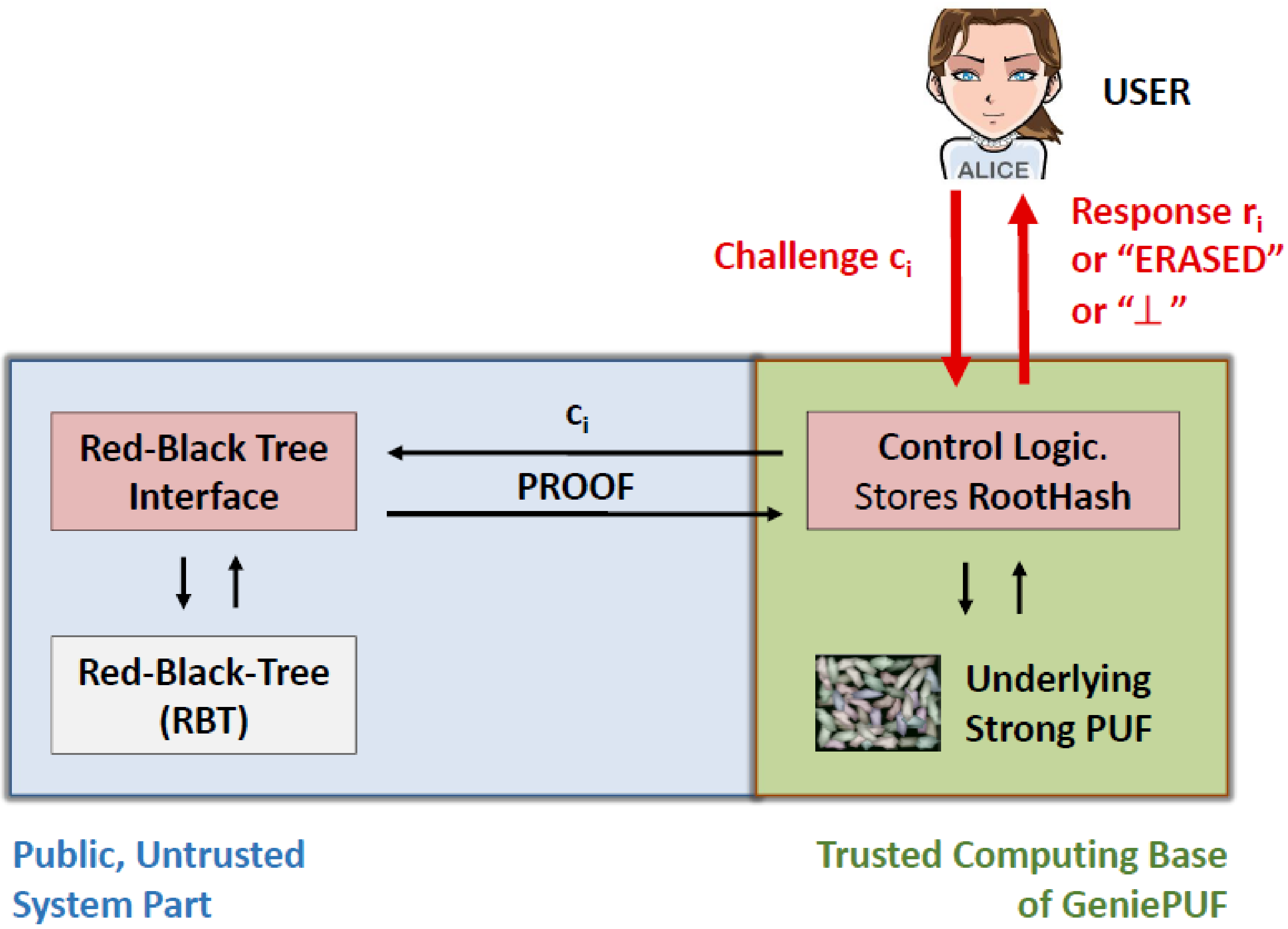
GeniePUF Architecture



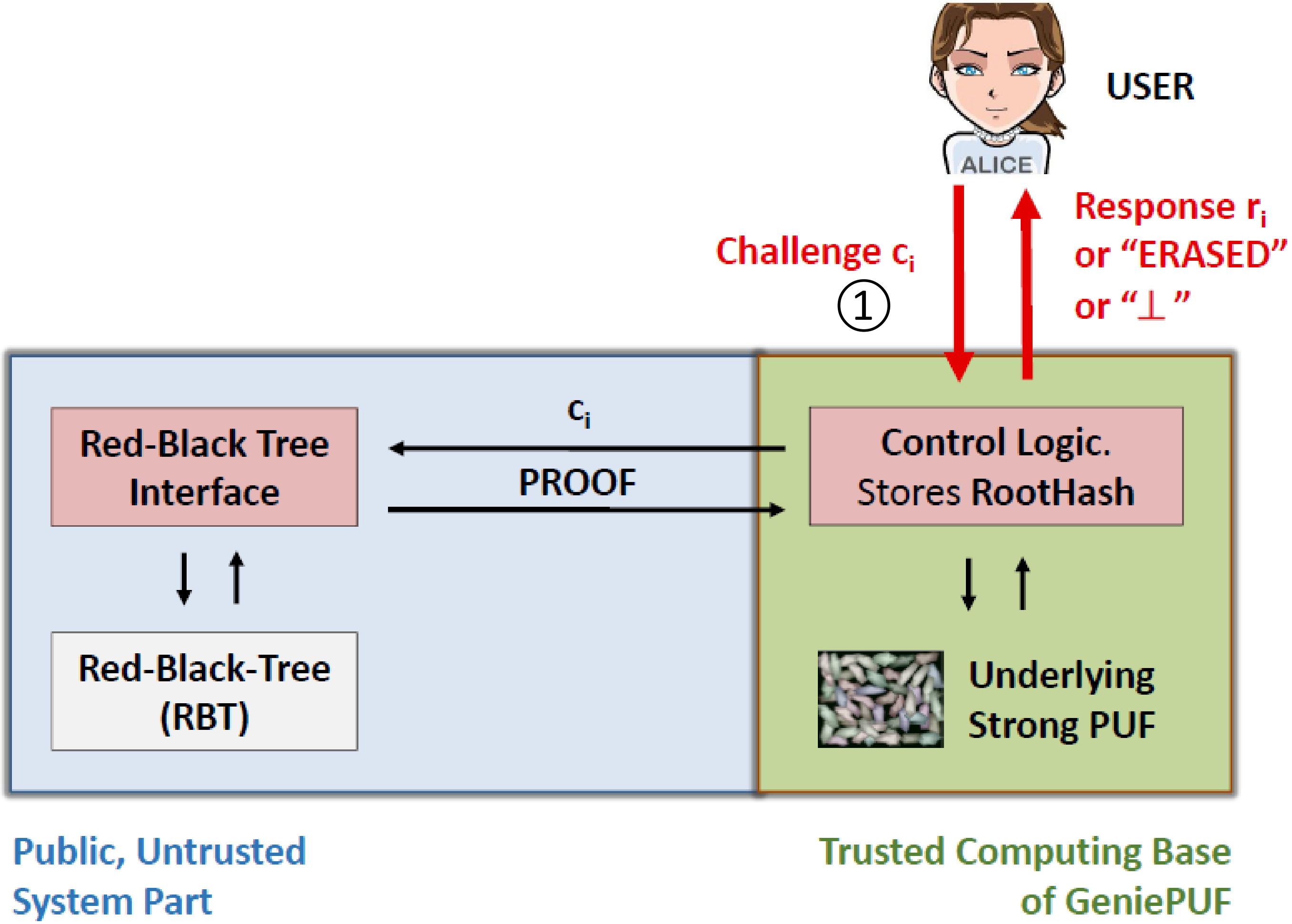
Public, Untrusted System Part (Software)

Trusted Computing Base (Hardware) of GeniePUF

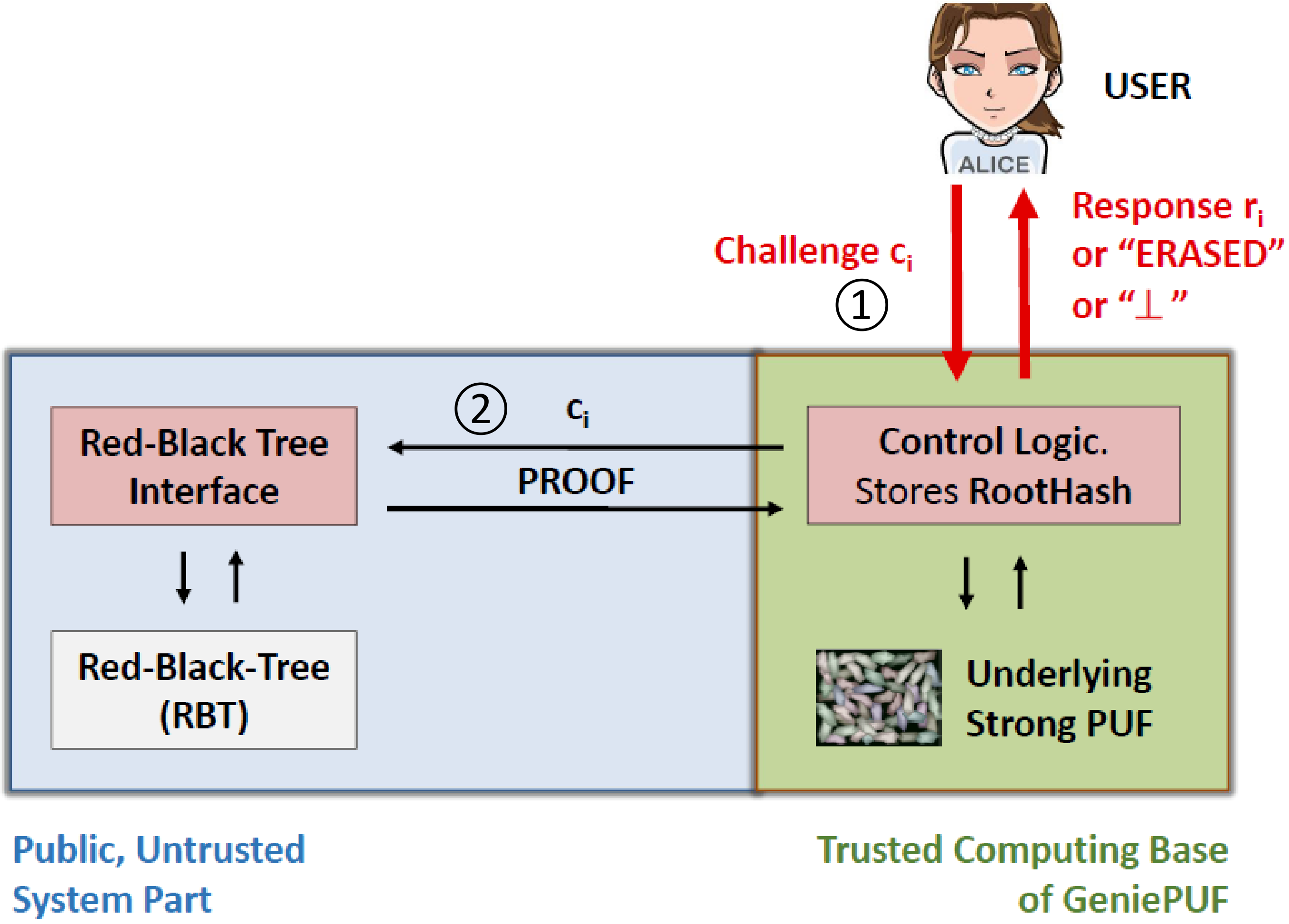
Read-Out Operation of Genie PUF



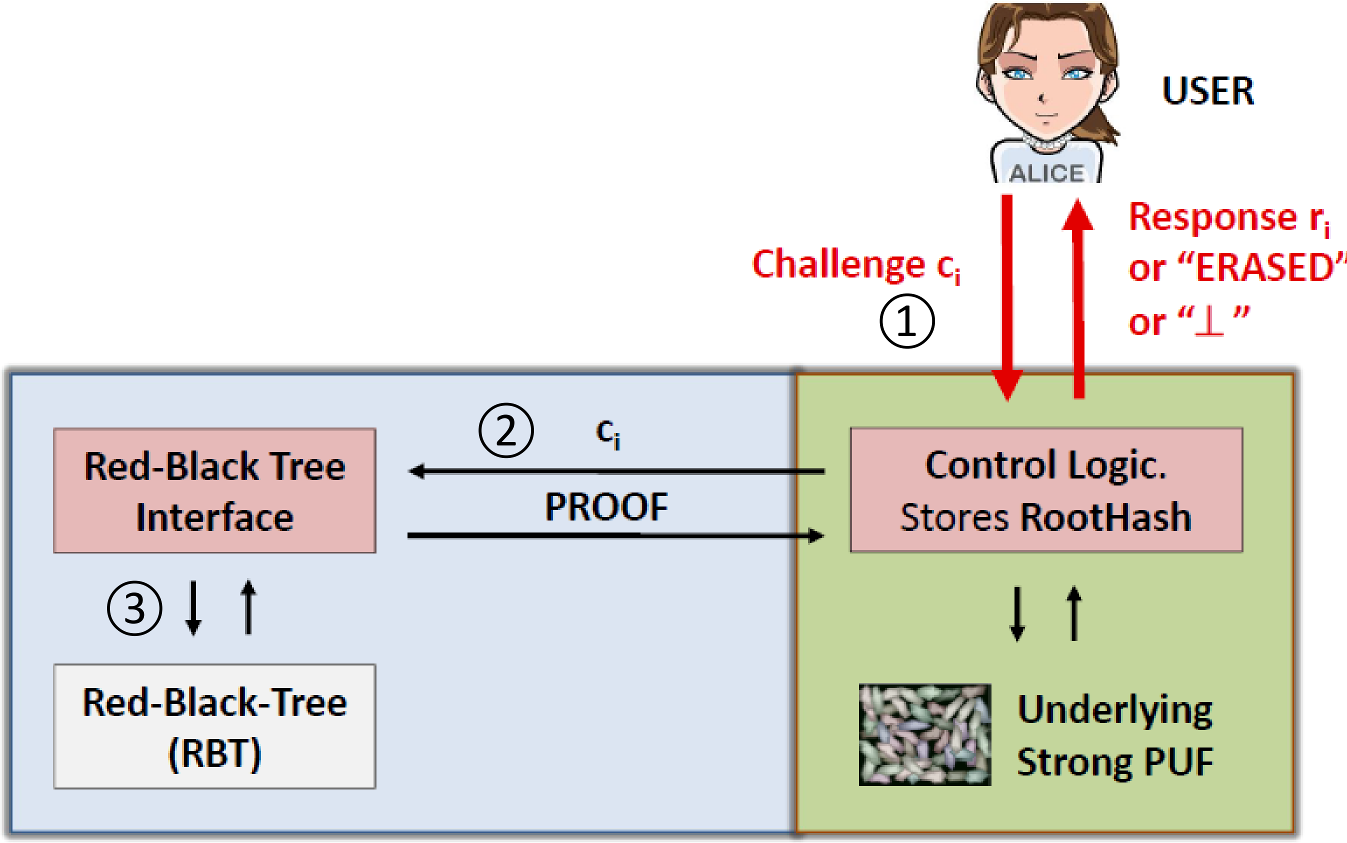
Read-Out Operation of Genie PUF



Read-Out Operation of Genie PUF



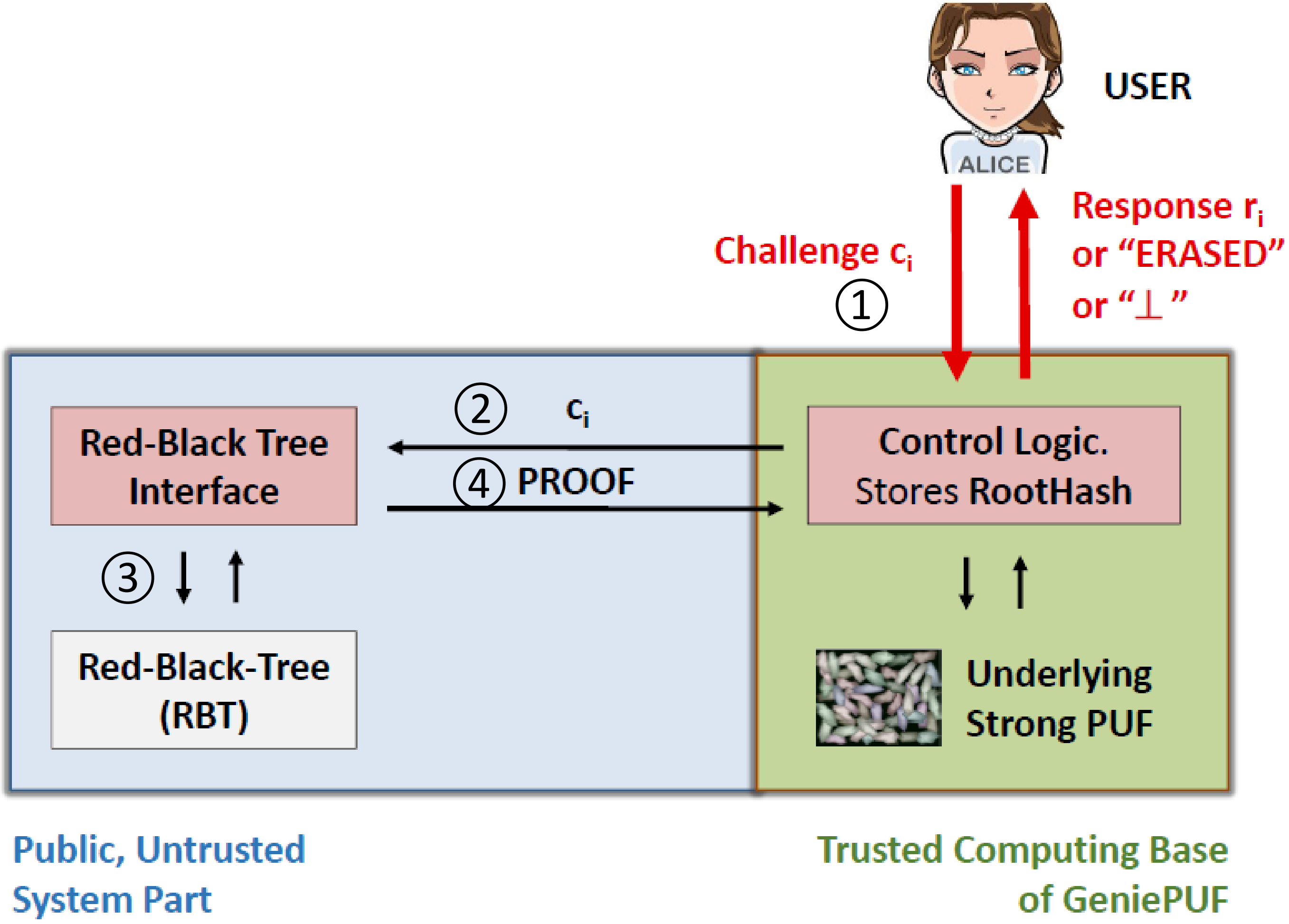
Read-Out Operation of Genie PUF



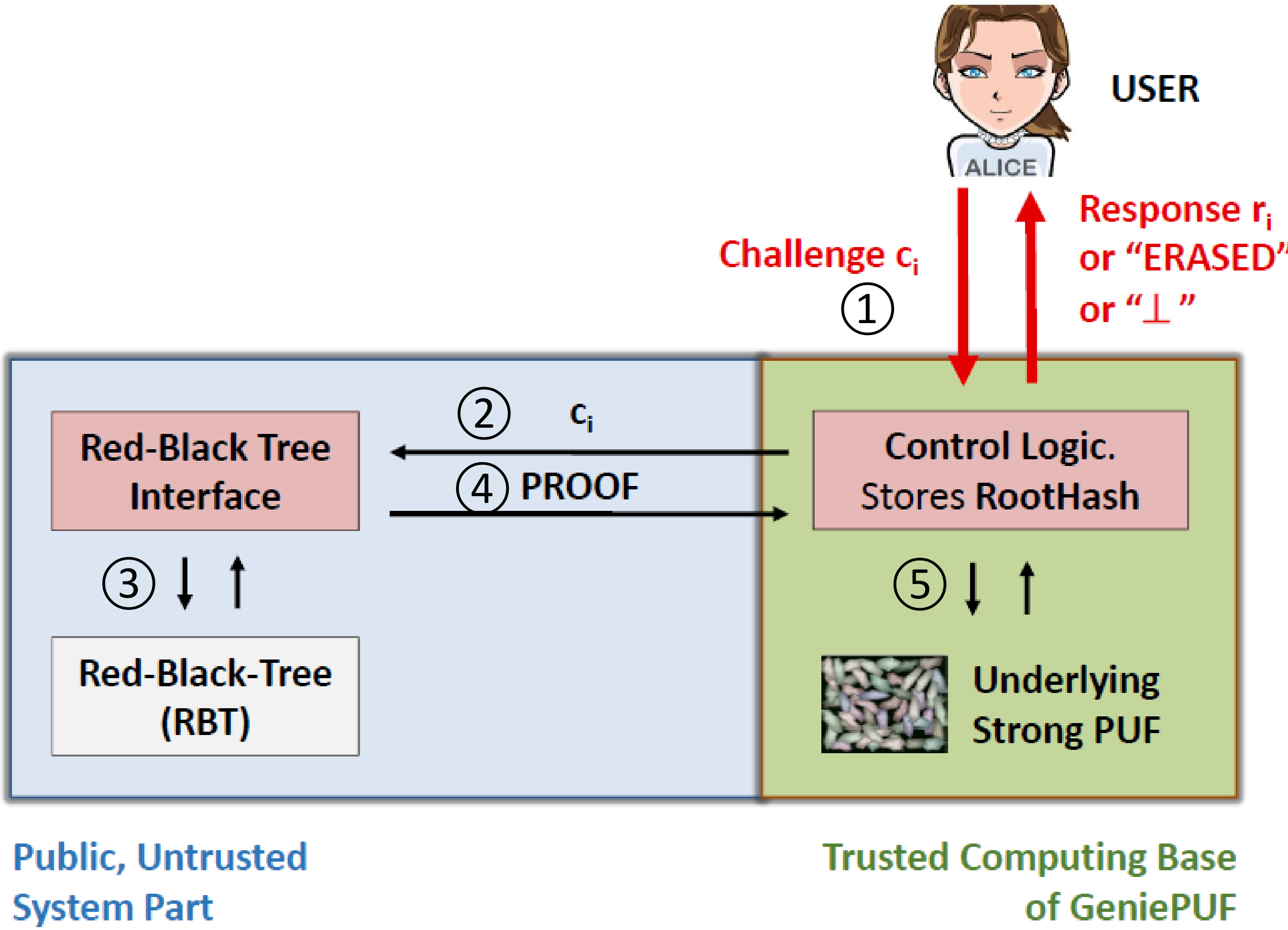
Public, Untrusted
System Part

Trusted Computing Base
of GeniePUF

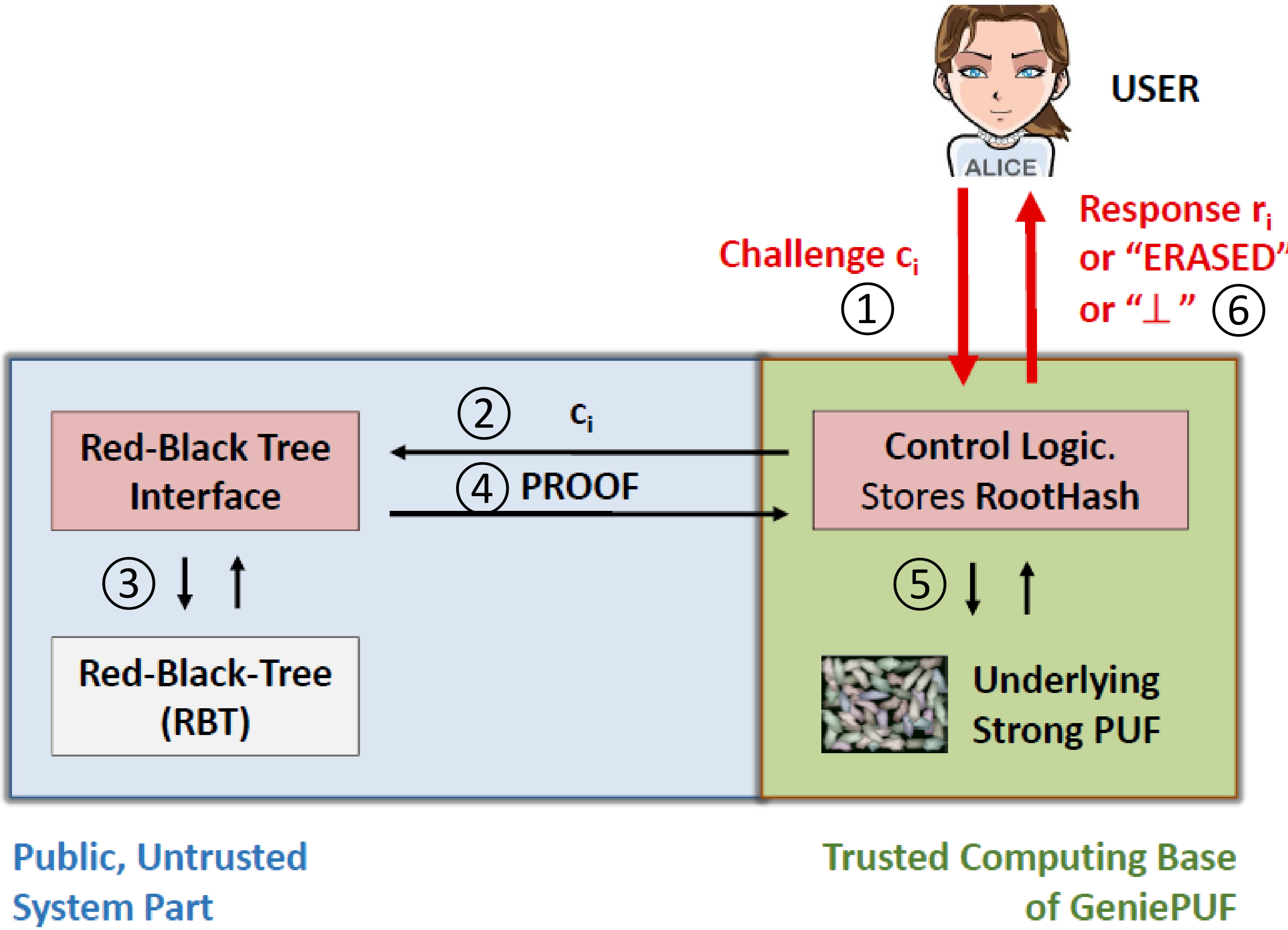
Read-Out Operation of Genie PUF



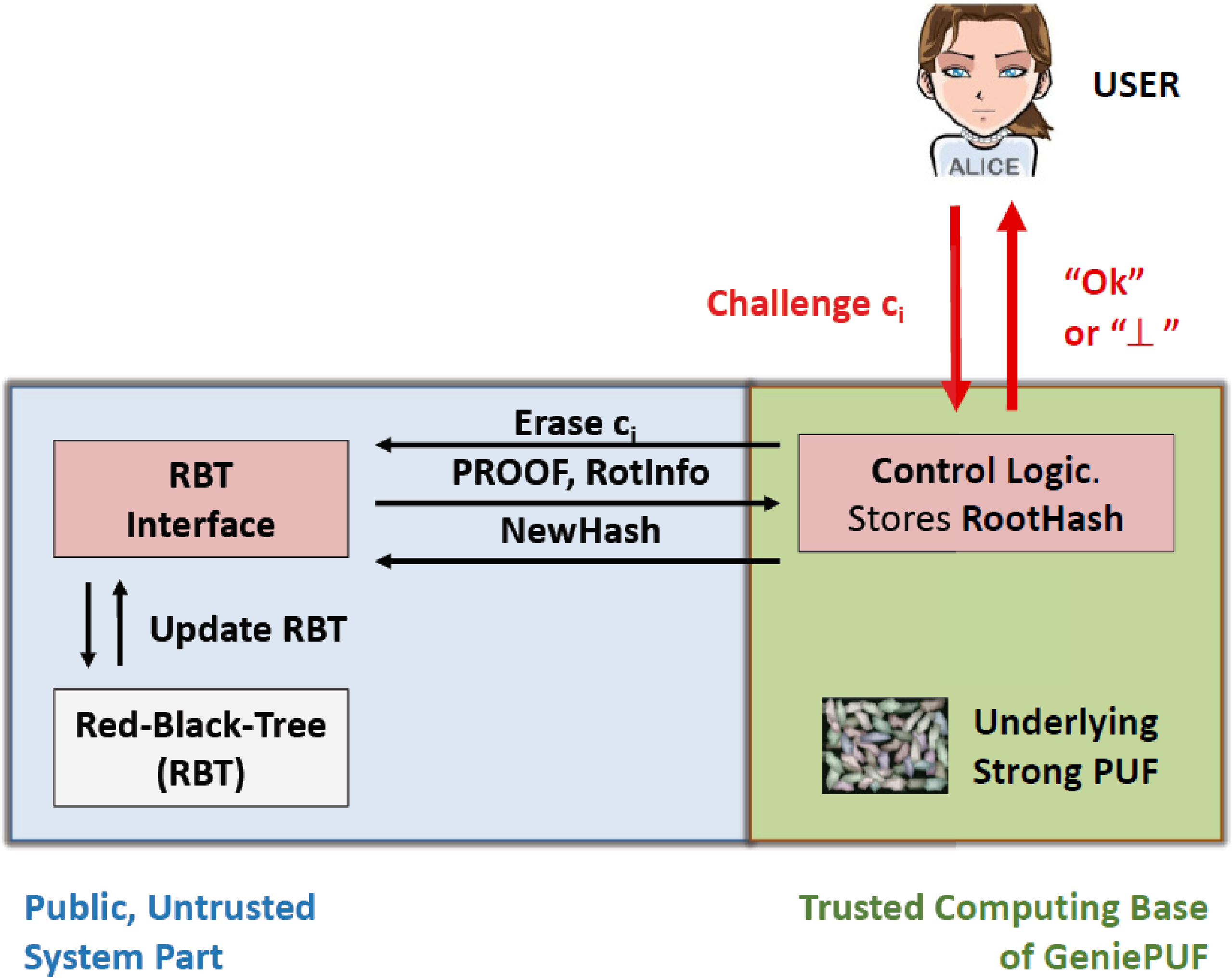
Read-Out Operation of Genie PUF



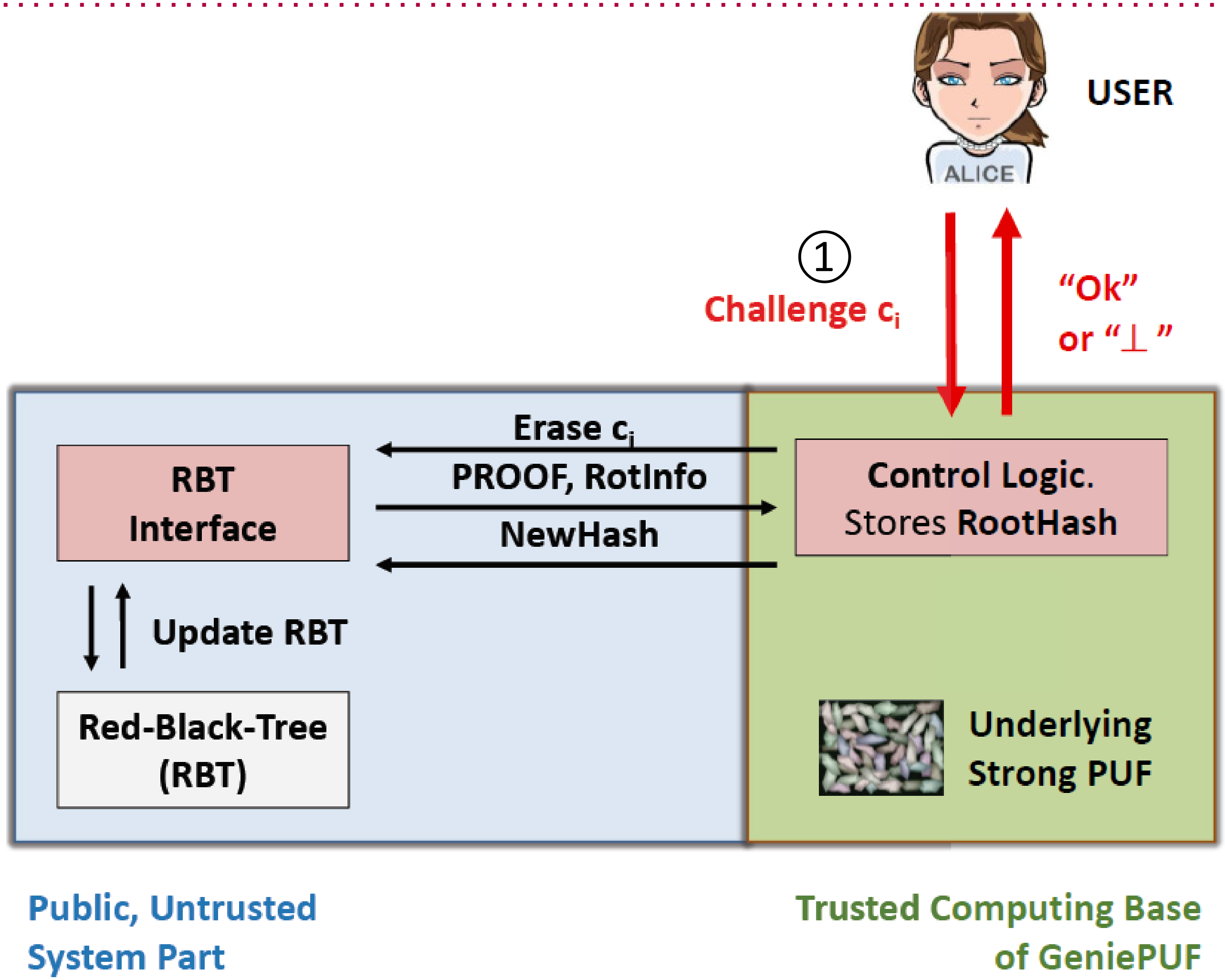
Read-Out Operation of Genie PUF



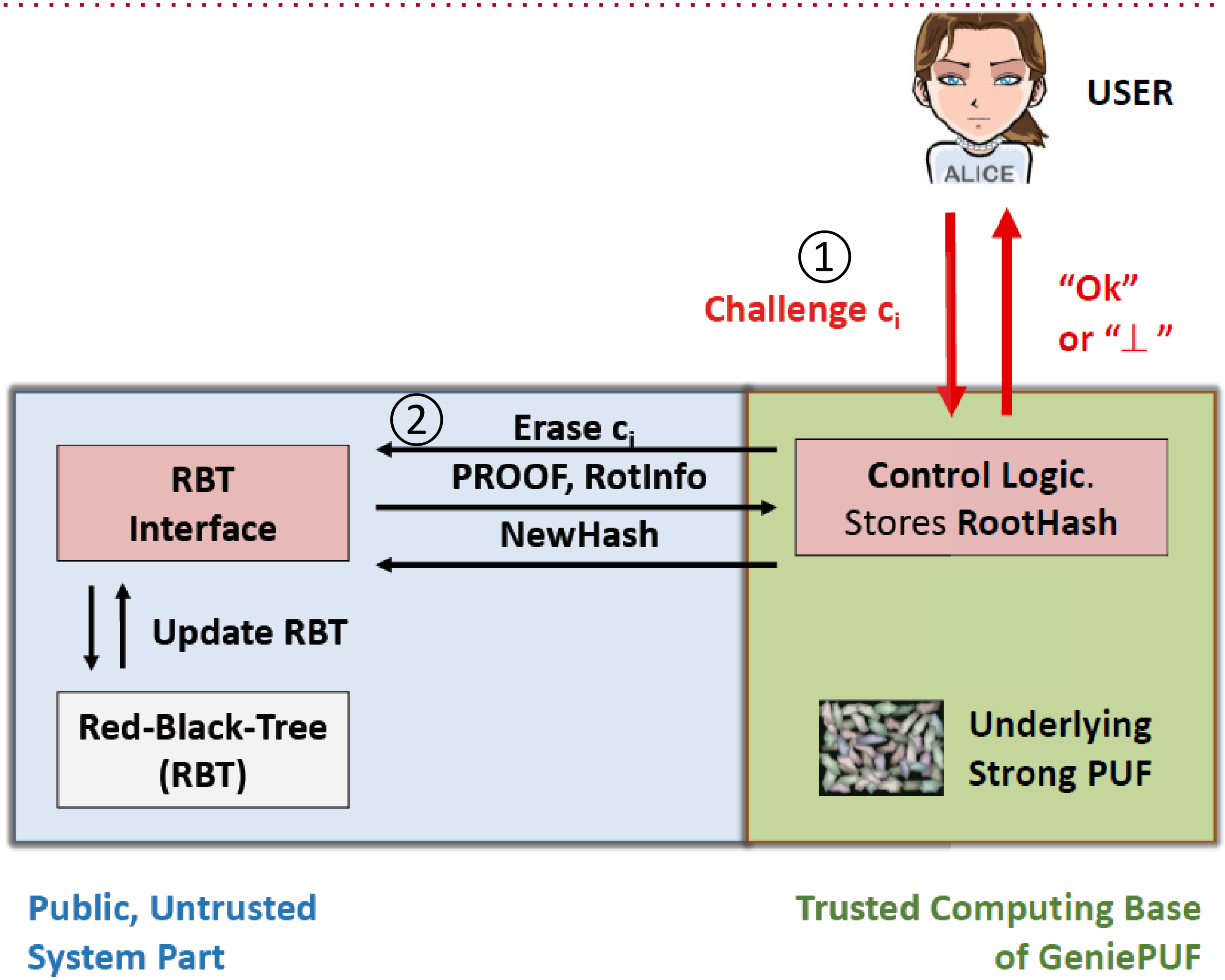
Erasure Operation of Genie PUF



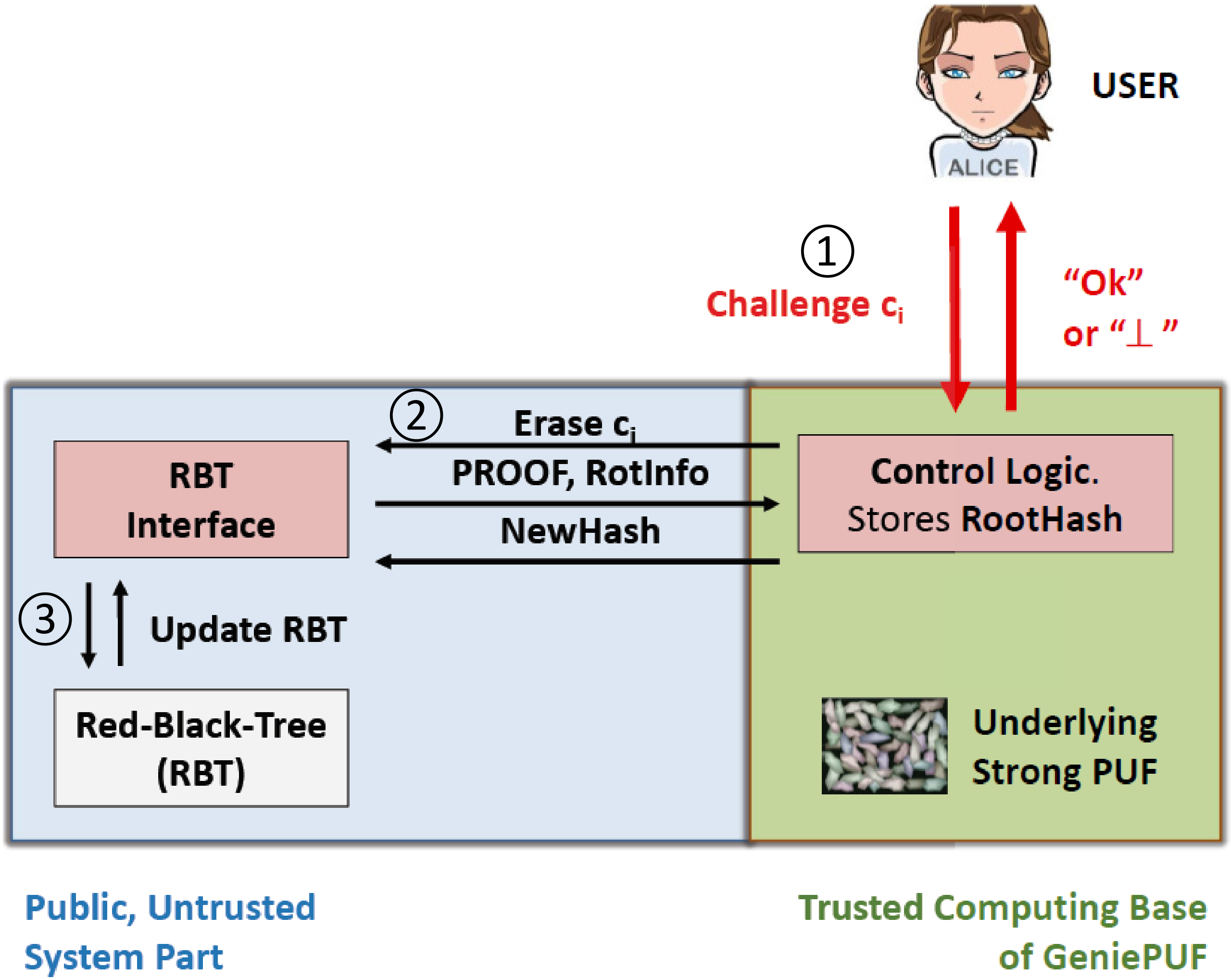
Erasure Operation of Genie PUF



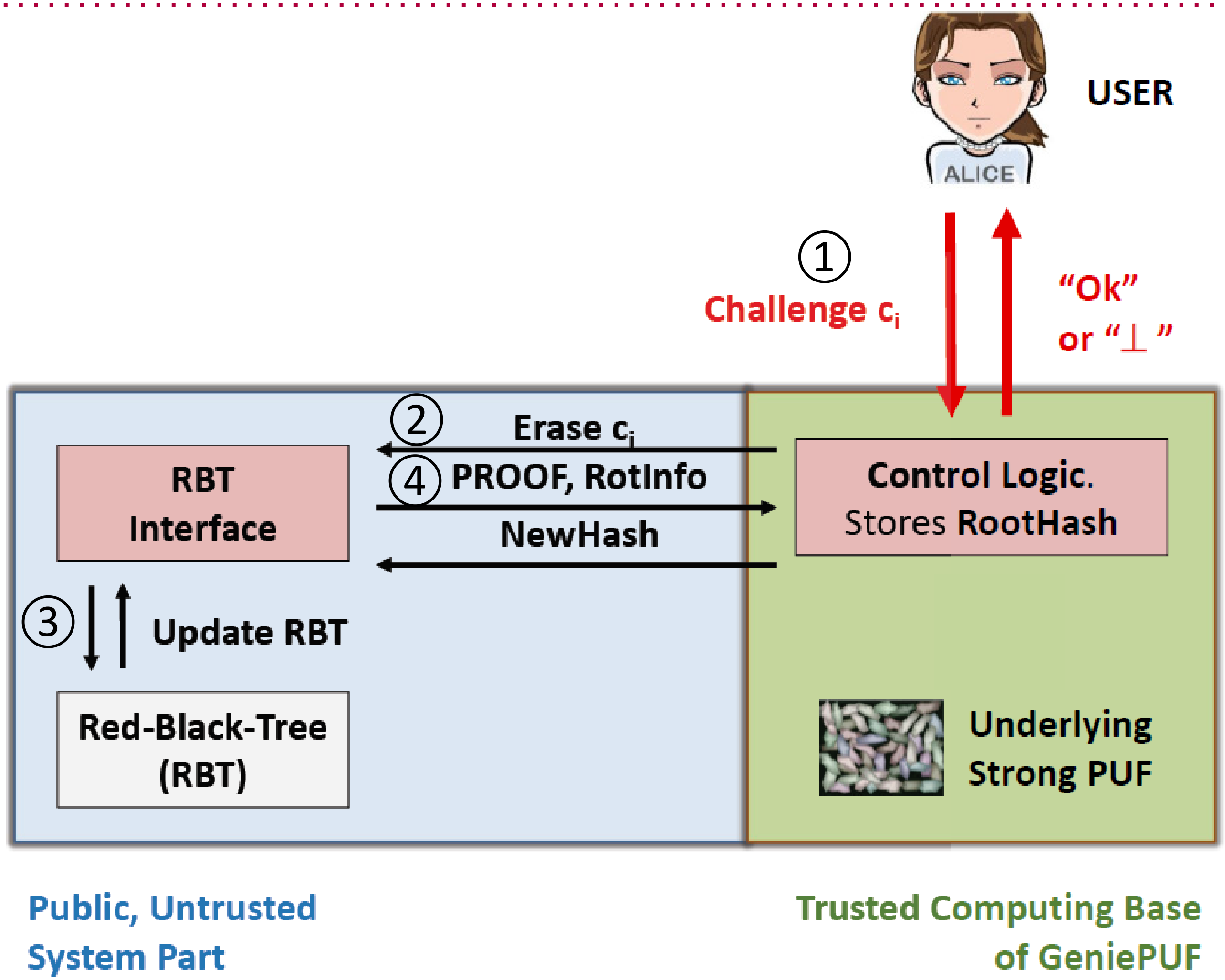
Erasure Operation of Genie PUF



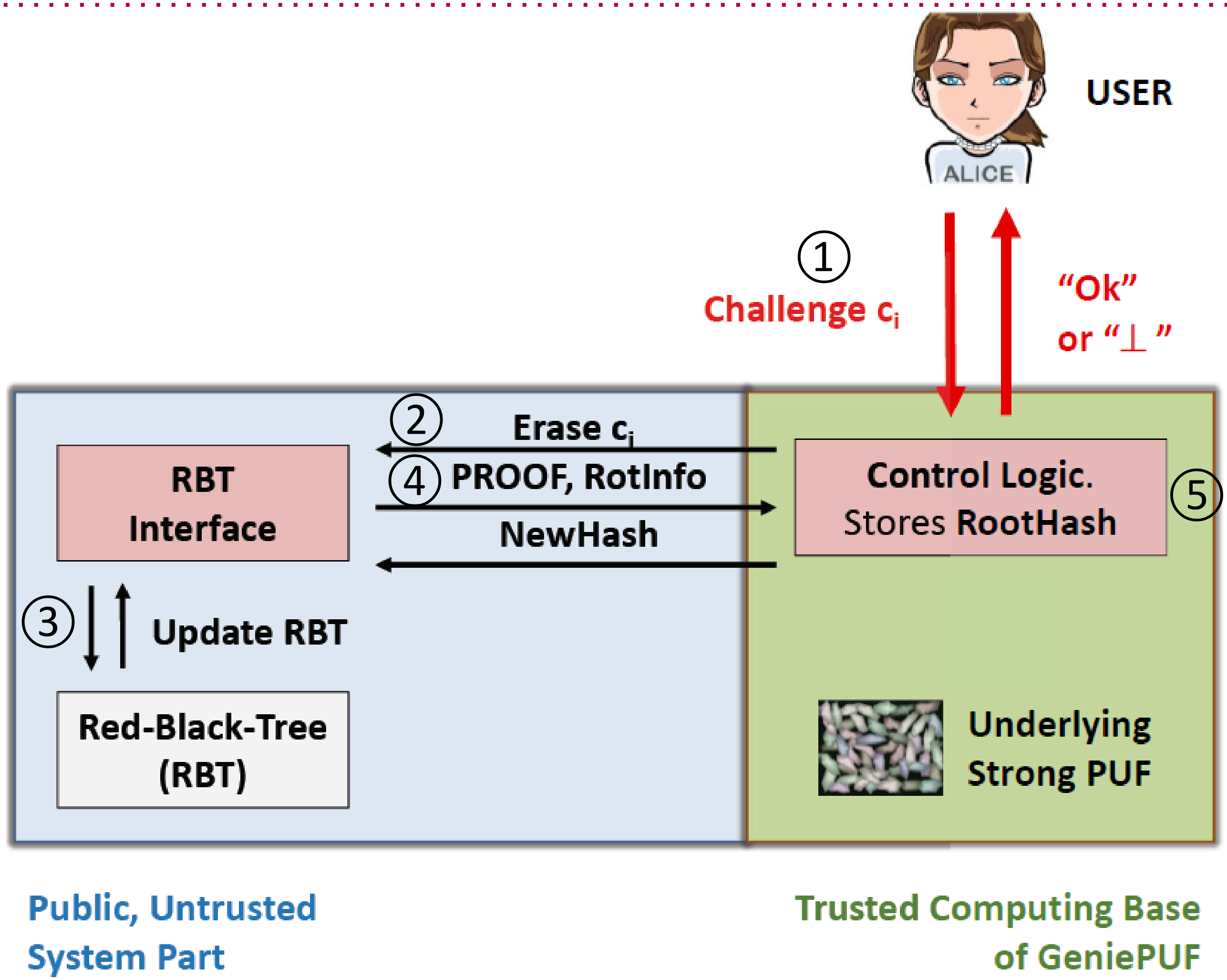
Erasure Operation of Genie PUF



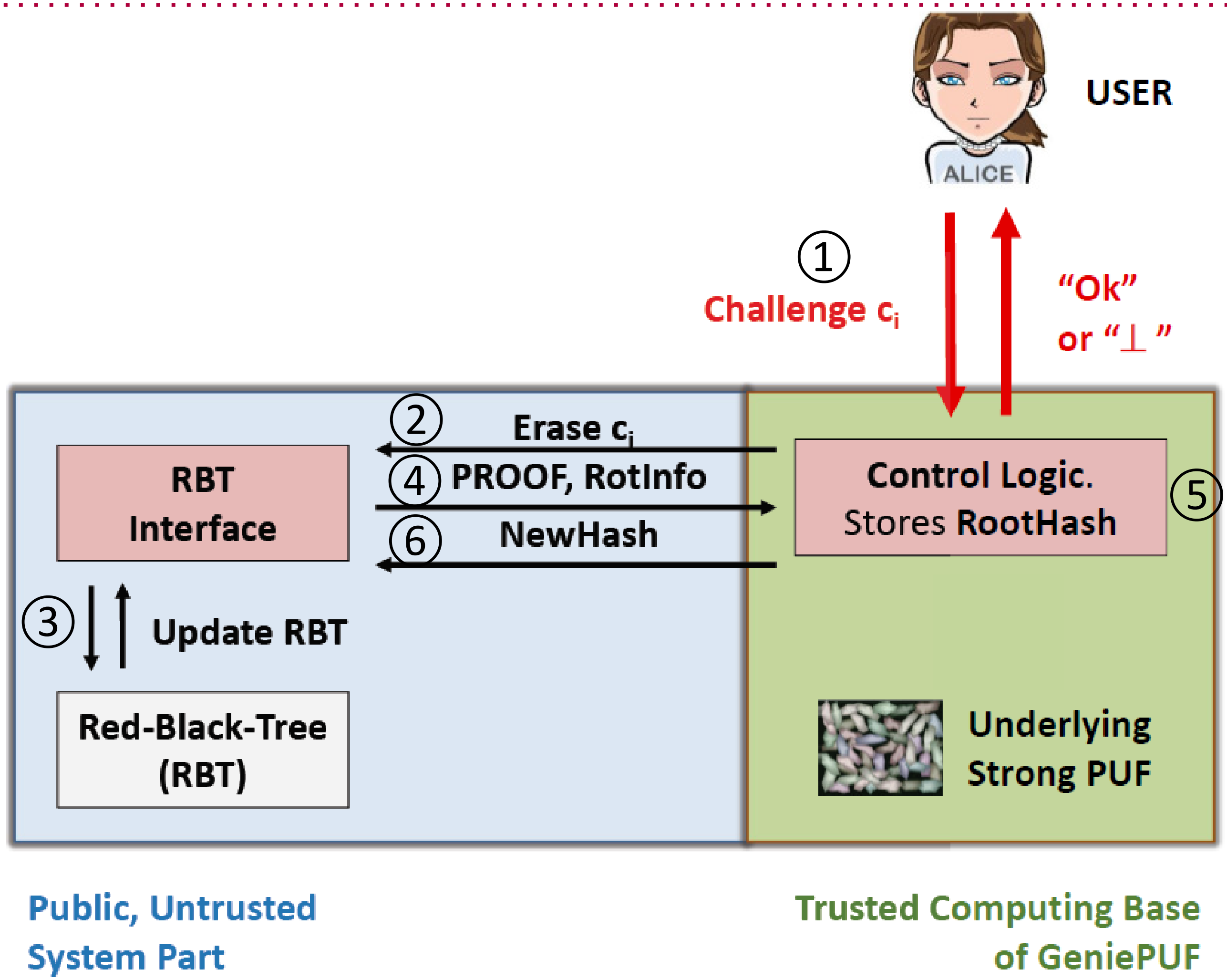
Erasure Operation of Genie PUF



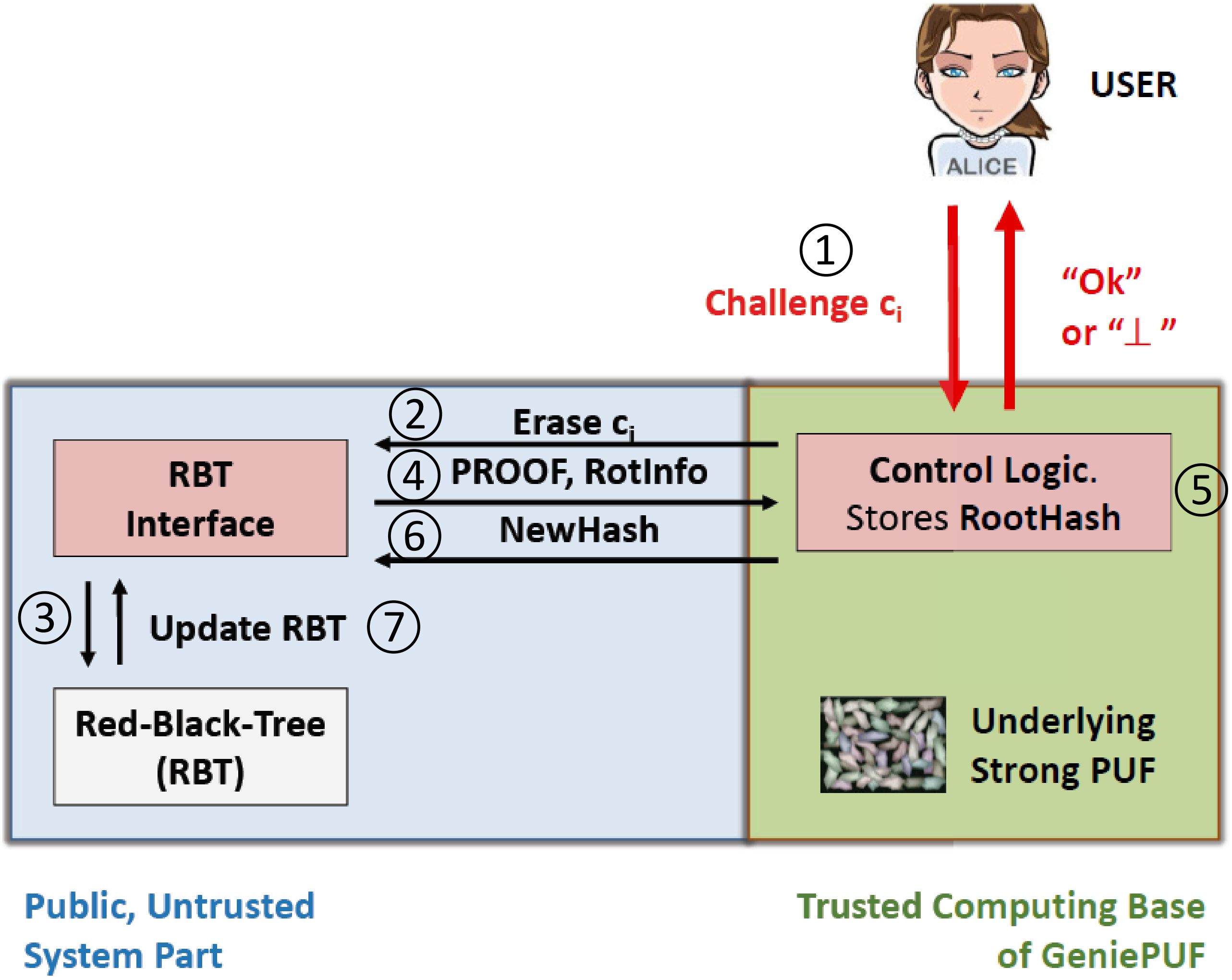
Erasure Operation of Genie PUF



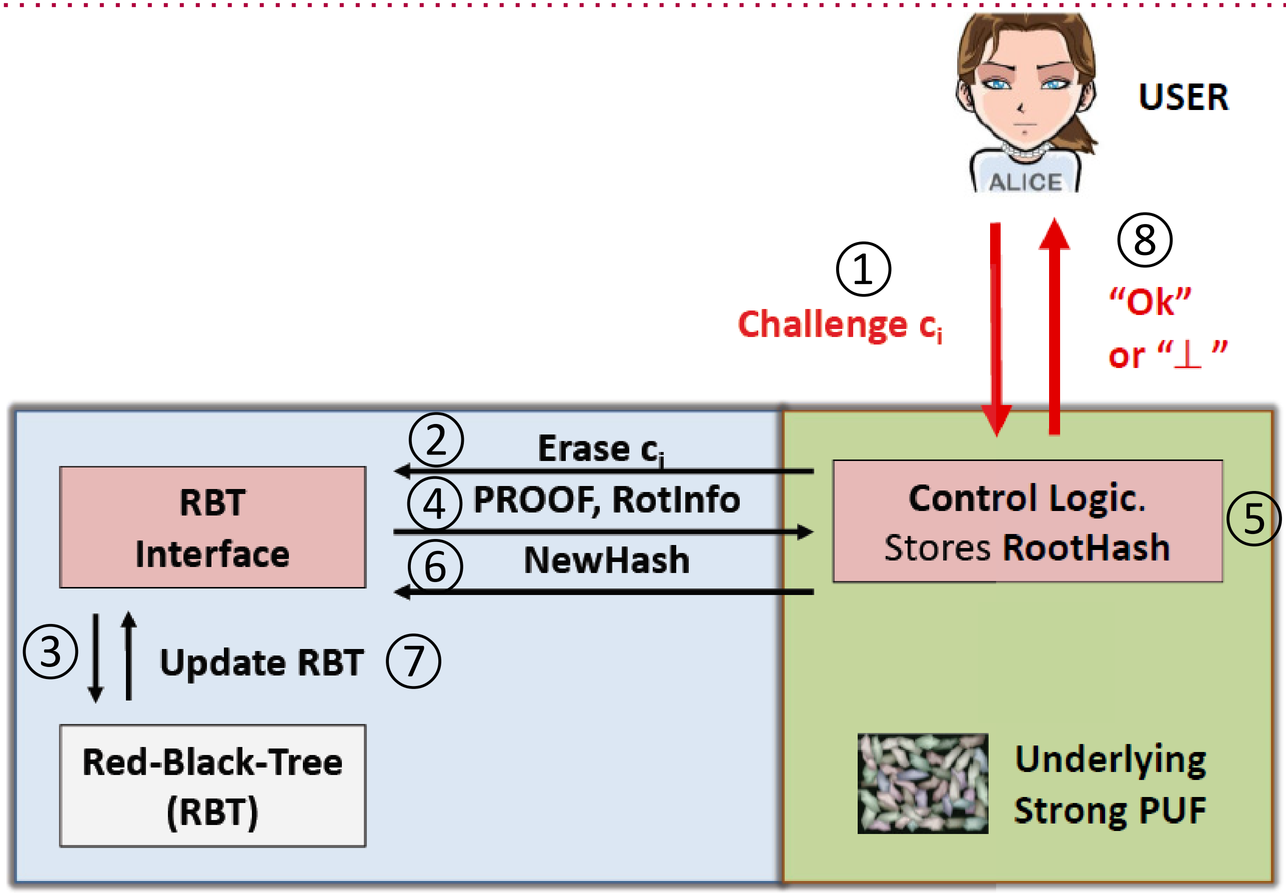
Erasure Operation of Genie PUF



Erasure Operation of Genie PUF



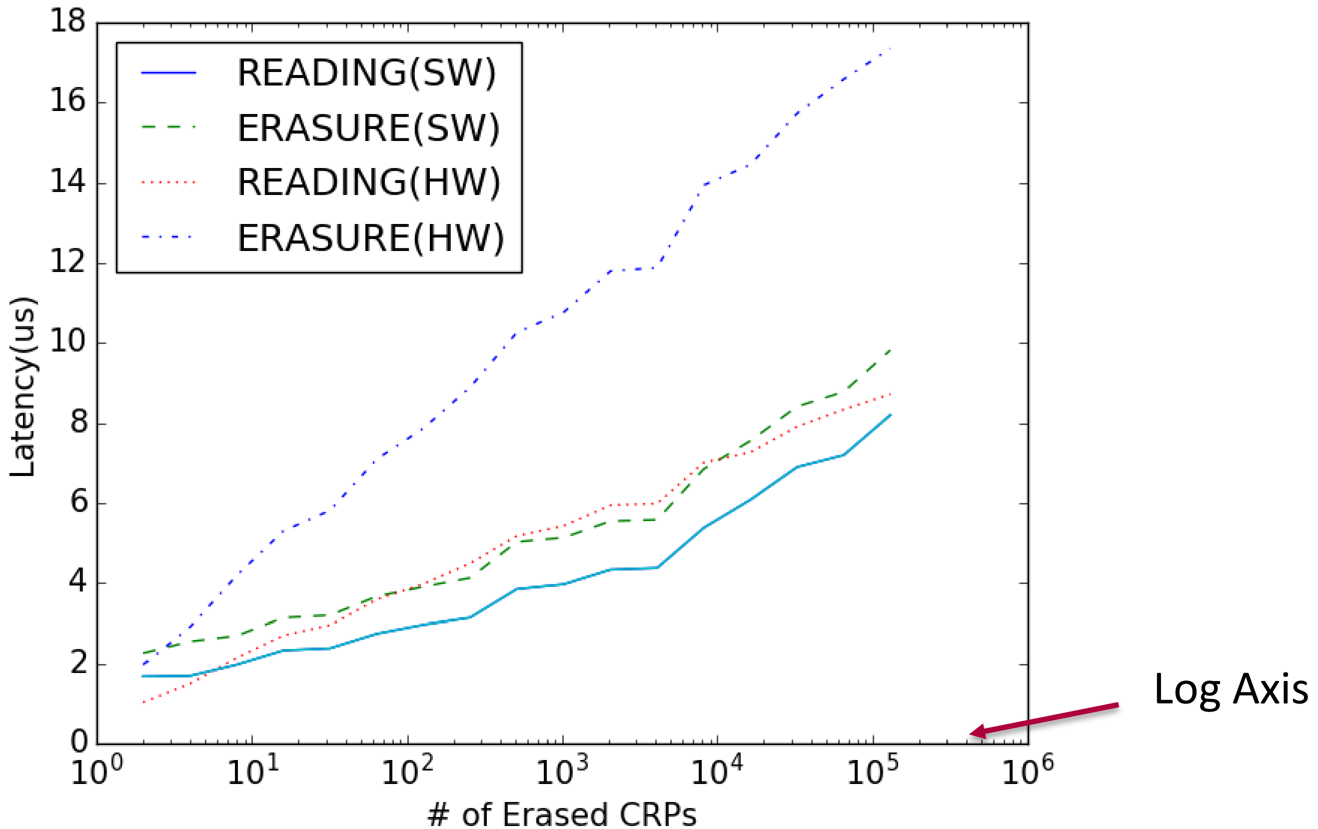
Erasure Operation of Genie PUF



Public, Untrusted
System Part

Trusted Computing Base
of GeniePUF

Performance Evaluation



- Implement the TCB on Zynq FPGA (HW) and the RB Tree Interface on Processor (SW)
- Latency grows logarithmically w.r.t. the number of erased challenges

Security Analysis

- Security Assumptions for Genie PUFs

1. Adversaries cannot circumvent the Control Logic (CL), applying their own challenges directly to the underlying Strong PUF, reading out the corresponding responses r_i.
2. Adversaries cannot modify the CL, for example such that it cannot correctly verify the validity of PROOF.
3. Adversaries may read the stored RootHash, but not modify it. It is public, but authentic.

A New Definitional Framework of PUFs

- Easily accessible, yet precise style PUF definition
- Parameterized Game-based PUF definition (ϵ, t_{att}, k)
- Intuition of Secure Erasable PUF Definition:

The security of an erasable PUF is measured by the upper bound ϵ of the accuracy of guessing one out of k randomly chosen CRPs by an attacker which takes time t_{att} for computation, physical actions, and k times game interactions with the challenger, where in each game interaction a randomly chosen CRP is erased.

Main Results of Formal Analysis

- Erasable PUFs are Strong PUFs
- Let P be a (k, t_{att}, ϵ)-secure Erasable PUF with respect to some adversary A. Then P is a (k, t_{att}, ϵ)-secure Strong PUF with respect to the same adversary A.

- The Security of Genie PUFs
- Let P be a PUF with challenge set C_P . Let A be an adversary for GeniePUF(P). Then GeniePUF(P) is $(k, t_{att}, \epsilon + \rho)$-secure Erasable PUF with respect to A, where ρ represents the collision probability of the used hash function.

Conclusion

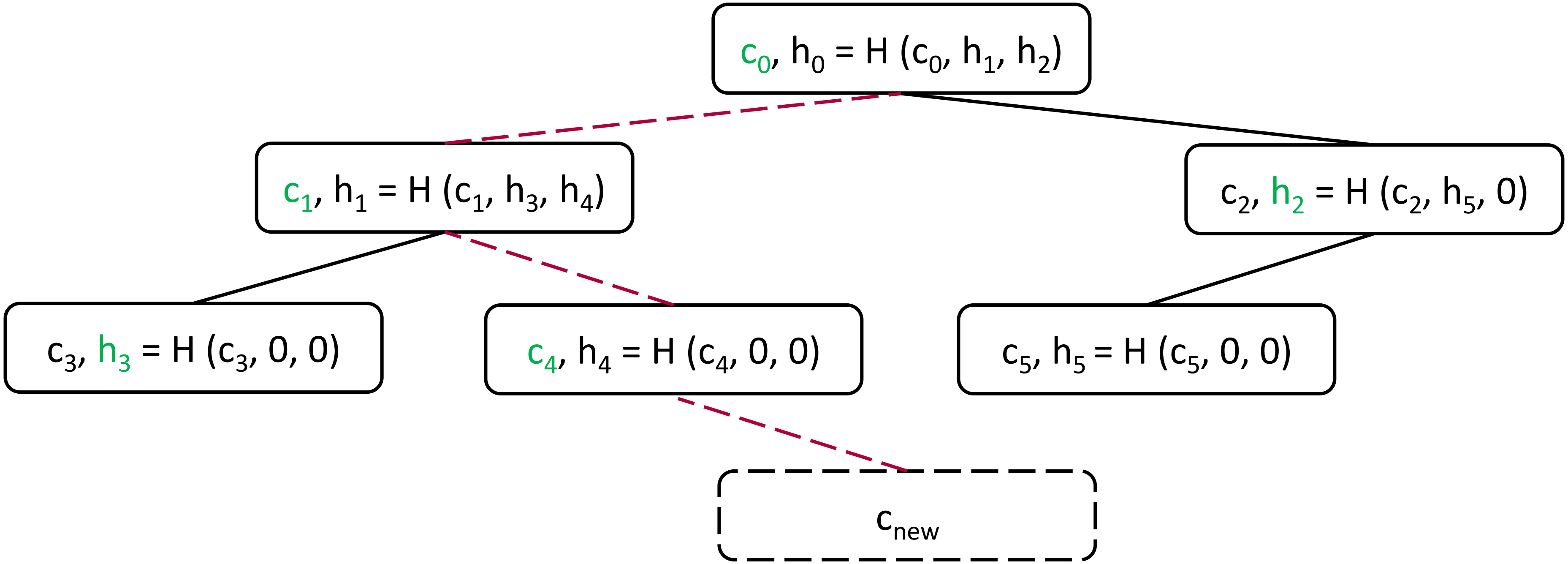
- Fixed the issue of PUF re-use model in PUF-based cryptographic protocols by using erasable PUFs.
- Introduced a generic erasable PUF design (Genie PUF) that can turn any strong PUFs to erasable PUFs.
- Proposed a rigorous, yet easily accessible definitional framework of PUF and proved our main theorems in the framework

Centrum Wiskunde & Informatica

Thank you for your attention!

Questions?

Authenticated Search Tree Proof Generation



1. Locate where the new challenge is supposed to be stored
2. Find a path from the new node for c_{new} to the root
3. Fetch all the challenge values and all sibling hash values to construct a proof of (non)-existence

Red-Black Tree Background

- Self-balancing Binary Search Tree
- Guarantee O(log N) worst-case search time with a tree of size N