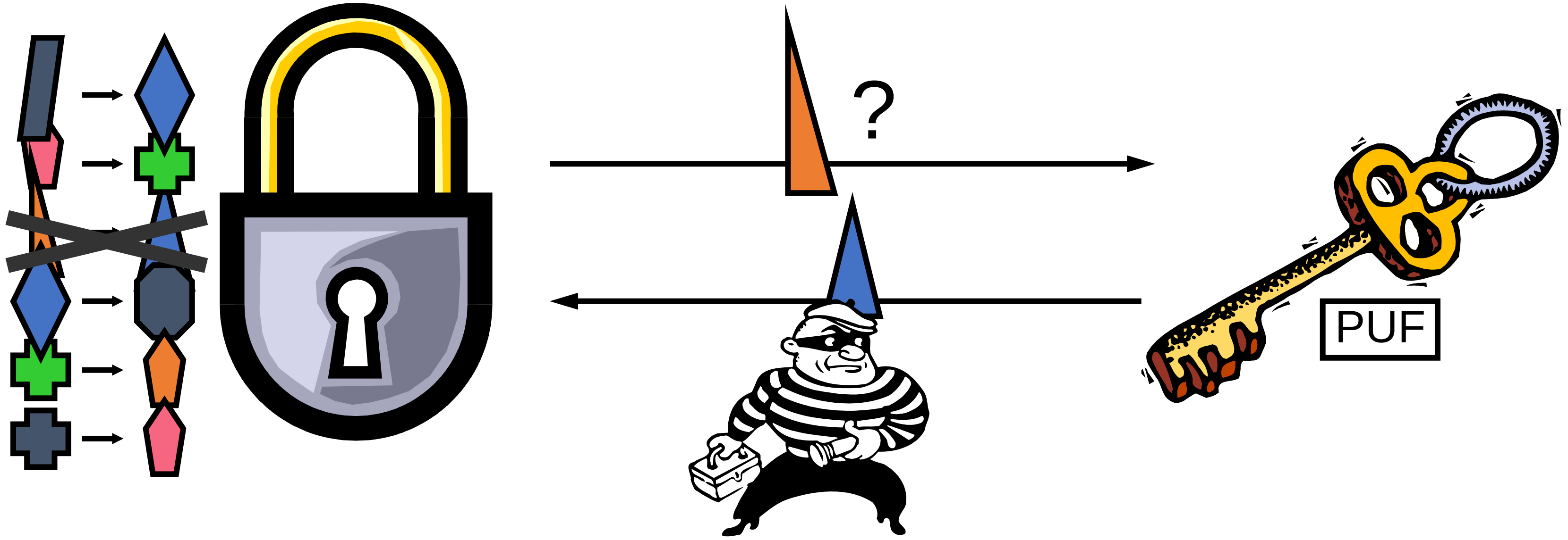
Using a PUF as an Unclonable Key

A Silicon PUF can be used as an unclonable key.

- **The lock has a database of challenge-response pairs.**
- **To open the lock, the key has to show that it knows the response to one or more challenges.**



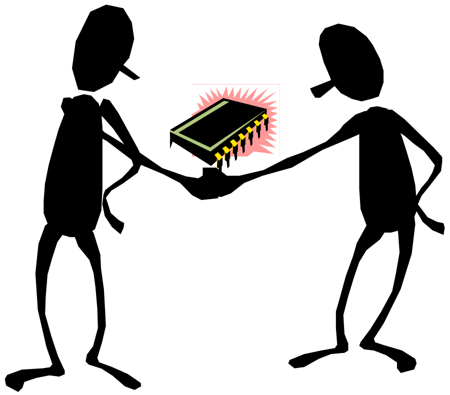
Applications

- **Anonymous Computation**

  Alice wants to run computations on Bob's computer, and wants to make sure that she is getting correct results. A certificate is returned with her results to show that they were correctly executed.

- **Software Licensing**

  Alice wants to sell Bob a program which will only run on Bob's chip (identified by a PUF). The program is copy-protected so it will not run on any other chip.



We can enable the above applications by trusting only a single-chip processor that contains a silicon PUF.

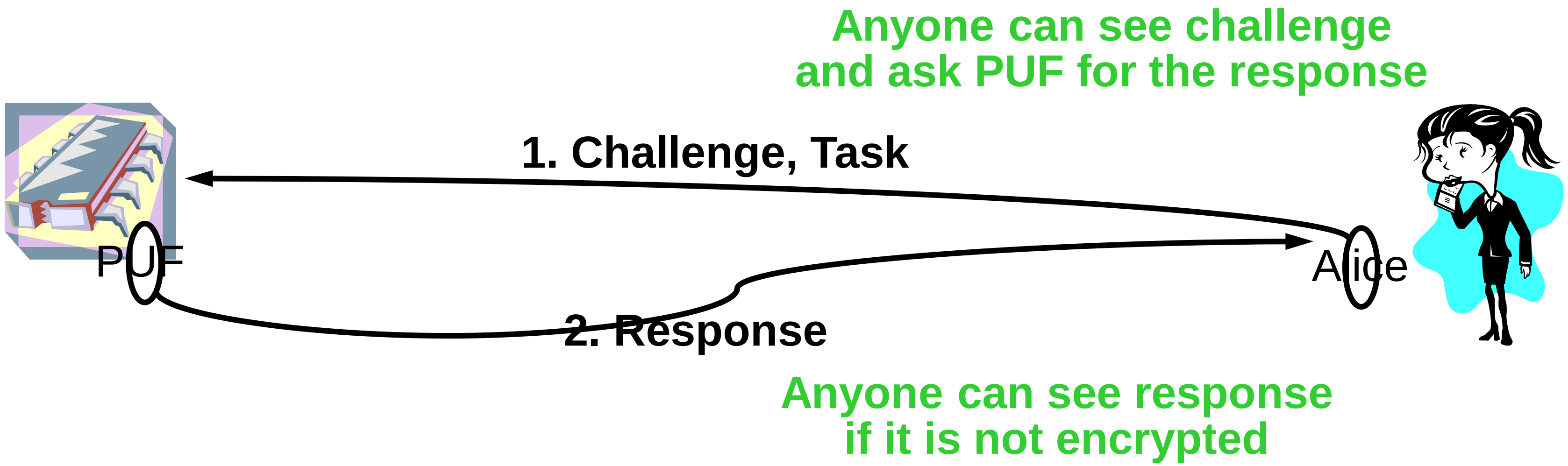
Sharing a Secret with a Silicon PUF

Alice has CRPs of different PUFs. At first the PUF will send its identification to Alice.

Suppose Alice wishes to share a secret with the silicon PUF

She has a challenge response pair that no one else knows, which can authenticate the PUF

She asks the PUF for the response to a challenge



Controlled Physical Random Functions (CPUFs)

- The PUF we used for authentication is bare. The information can be leaked during unsecured communication.
- The man-in-the middle attacker could get information of challenge-response pairs from the channel. To prevent the attack, the man in the middle must be prevented from finding out the response.
- So, we can add an extra layer to the PUFs so that attacker does not get any information of the response.
- PUF with an additional secured layer is known as controlled PUF (CPUF)
- CPUFs can be used to establish a secret between a physical device and remote user. So, CPUFs are PUFs that only can be accessed via an algorithm.
- Hash program is used as extra layer, which increases the level of the security.

Sharing a Secret with a Silicon PUF

- A **hash function** is any algorithm or subroutine that maps large data sets of variable length, called *keys*, to smaller data sets of a fixed length.
- For example, a person's name, having a variable length, could be hashed to a single integer.
- Alice knows the response because she had the challenge-response pair, but the man in the middle doesn’t know response since he can’t predict PUF(challenge).
- So Alice can compute secret

Cryptographic Hash Function

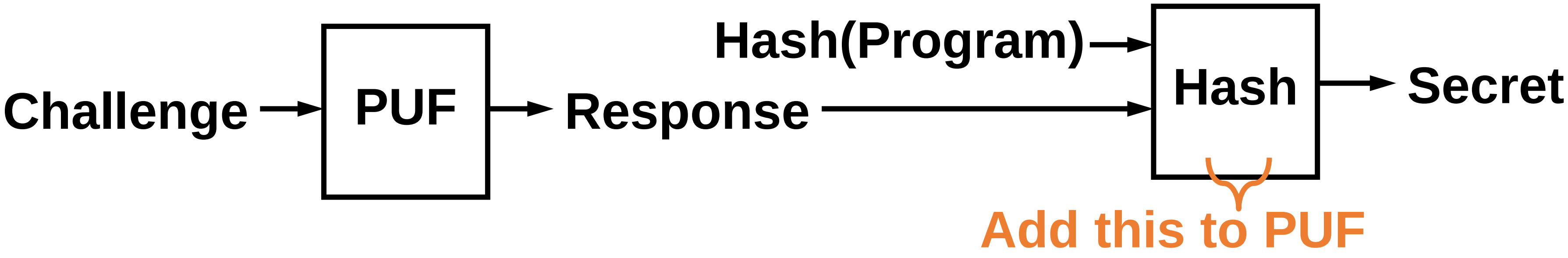
- Crypto hash function $h(x)$ must provide
  - **Compression** — output length is small
  - **Efficiency** — $h(x)$ easy to compute for any x
  - **One-way** — given a value y it is infeasible to find an x such that $h(x) = y$
  - **Weak collision resistance** — given x and $h(x)$, infeasible to find $y \neq x$ such that $h(y) = h(x)$
  - **Strong collision resistance** — infeasible to find any x and y, with $x \neq y$ such that $h(x) = h(y)$

Restricting Access to the PUF

- To prevent the attack, the man in the middle must be prevented from finding out the response.
- Alice's program must be able to establish a shared secret with the PUF, the attacker's program must not be able to get the secret.

⇒ Combine response with hash of program.

- **The PUF can only be accessed via the GetSecret function:**



Getting a Challenge-Response Pair

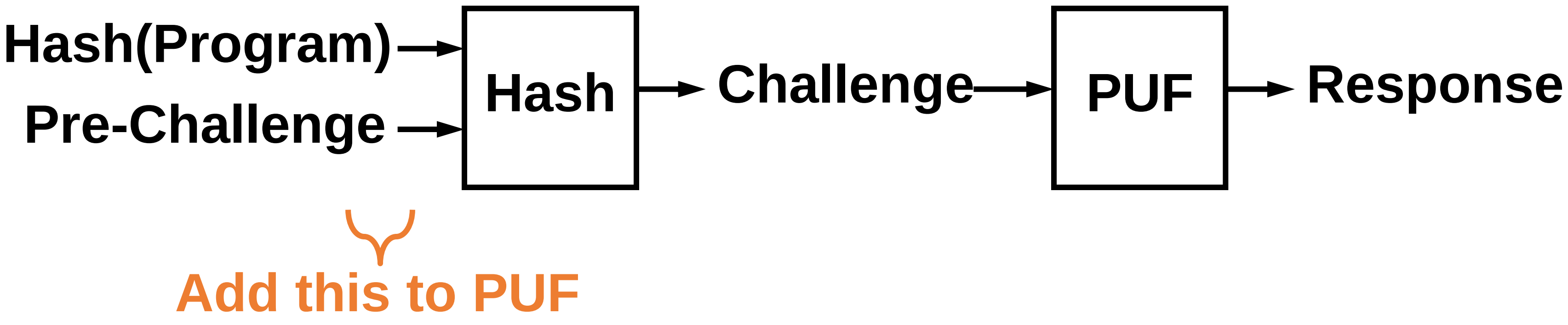
- Now Alice can use a Challenge-Response pair to generate a shared secret with the PUF equipped device.

- But Alice can't get a Challenge-Response pair in the first place since the PUF never releases responses directly.

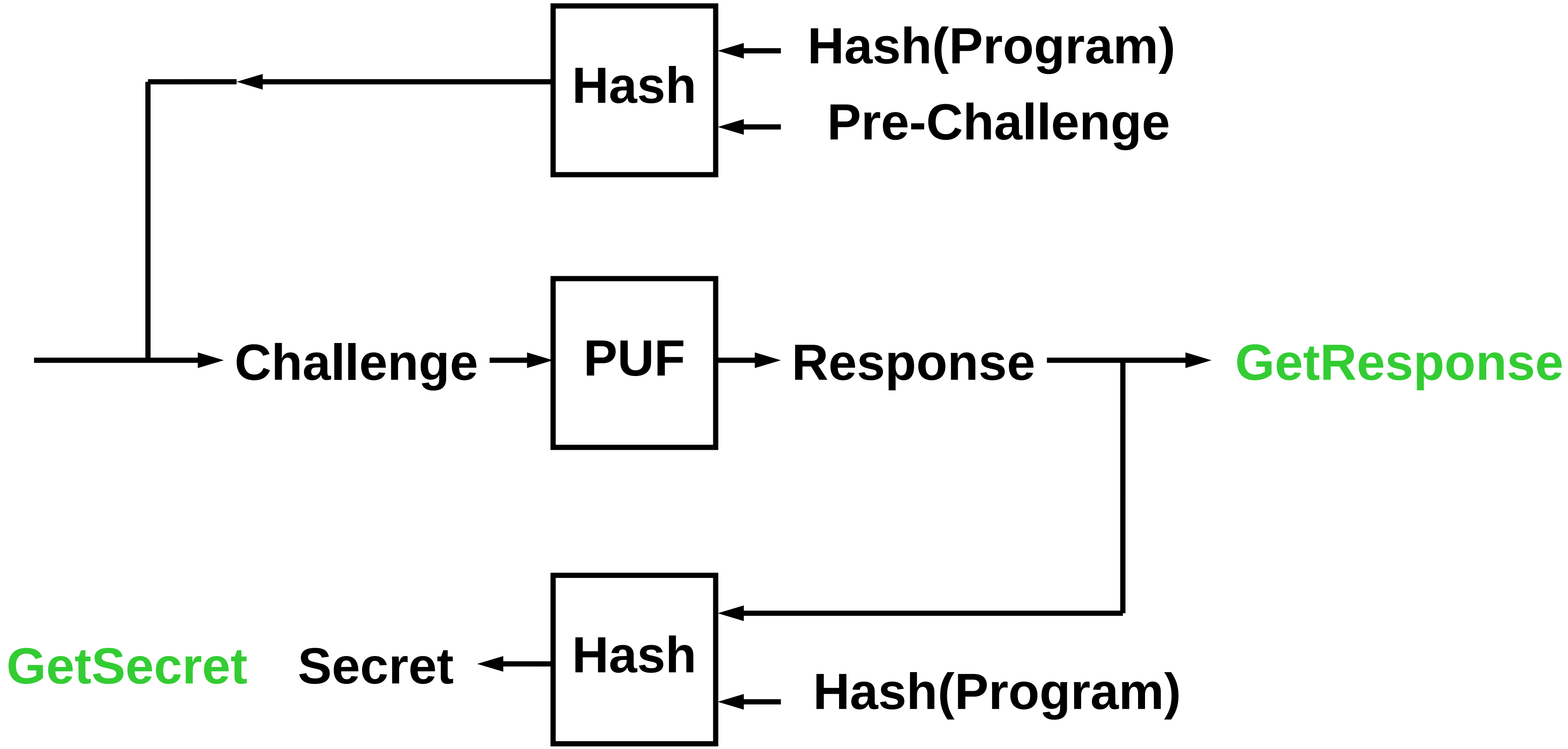
$\Rightarrow$ An extra function that can return responses is needed.

Getting a Challenge-Response Pair – 2

- Let Alice use a Pre-Challenge.
- Use program hash to prevent eavesdroppers from using the pre-challenge.
- The PUF has a GetResponse function
- **Bootstrapping – need to be in direct contact with the PUF. Else if PUF is remote, Alice already has a challenge-response pair, and uses the old response to encrypt the new response.**



Controlled PUF Implementation



Challenge-Response Pair Management: Bootstrapping

- When a Controlled PUF (CPUF) has just been produced, the manufacturer wants to generate a challenge-response pair.
- Ecode has been encrypted with Secret by Manufacturer Secret is known to the manufacturer because he knows Response to Challenge and can compute
- Secret = Hash(Hash(Program), Response).
- Adversary cannot determine Secret because he does not know Response or Pre-Challenge. If adversary tries a different program, a different secret will be generated because Hash(Program) is different.
- Pre-challenge will be thrown away after using to increase the security level.

Software Licensing

Program (Ecode, Challenge)

Secret = GetSecret(Challenge)

Code = Decrypt(Ecode, Secret)

Run Code

Hash(Program)

Ecode has been encrypted with Secret by Manufacturer

Secret is known to the manufacturer because he knows Response to Challenge and can compute

Secret = Hash(Hash(Program), Response)

Adversary cannot determine Secret because he does not know Response or Pre-Challenge

If adversary tries a different program, a different secret will be generated because Hash(Program) is different

Summary

- PUFs provide secret “key” and CPUFs enable sharing a secret with a hardware device
- CPUFs are not susceptible to model-building attack if we assume physical attacks cannot discover the PUF response
  - Control protects PUF by obfuscating response, and PUF protects the control from attacks by “covering up” the control logic
  - Shared secrets are volatile