The Shift PUF: Technique for Squaring the Machine Learning Complexity of Arbiter-based PUFs

Yi Tang 1  Donghang Wu 2  Yongzhi Cao 2  Marian Margraf 3

1University of Michigan
2Peking University
3Freie Universität Berlin

CASES 2020 WiP
Physically Unclonable Functions (PUFs)

- Hardware cryptographic identification primitive;
- Exploiting inevitable manufacturing variations;
- Hence “physically unclonable.”

A PUF Instance is identified by its behavior as a (probabilistic) mapping from inputs (challenges) to outputs (responses).
Physically Unclonable Functions (PUFs)

- Hardware cryptographic identification primitive;
- Exploiting inevitable manufacturing variations;
- Hence “physically unclonable.”

A PUF Instance is identified by its behavior as a (probabilistic) mapping from inputs (challenges) to outputs (responses).
Arbiter PUFs (APUFs)

1. Challenge decides the paths: either parallel or crossing.
2. Two signals are triggered simultaneously and propagate along the decided paths.
3. *Arbiter* judges the race and yields the result as response.

The inevitable manufacturing variations of the signal delays lead to unique challenge-response behaviors.

---

Arbiter PUFs (APUFs)

1 Challenge decides the paths: either parallel or crossing.

Two signals are triggered simultaneously and propagate along the decided paths.

*Arbiter* judges the race and yields the result as response.

The inevitable manufacturing variations of the signal delays lead to unique challenge-response behaviors.

---

1 Figure from Georg T. Becker, “The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs”, 2015.
Desired Properties of a PUF design

- **Being lightweight**: low time and circuit complexities of the hardware;
- **Being strong**: huge challenge space, thus huge PUF instance space;
- **Reliability**: the same challenge results in the same response w.h.p.;
- **Security**: hard to predict with high accuracy the challenge-response behavior given reasonable amount of information.
Security of APUFs

- lightweight, strong, and reliable.
- However insecure. Challenge-response behavior $r(c)$ of $m$-bit APUF:

$$r(c) = [\Delta(c) \geq 0] , \quad \Delta(c) = w^\top p ,$$

- $w$: $(m + 1)$-d vector that is only a function of the signal delays in the APUF instance,
- $p$ (parity): $(m + 1)$-d vector that is only a function of the challenge $c$.
- Linear classification model; easy to learn given reasonable number of challenge-response pairs (CRPs).
Security of APUFs

- Lightweight, strong, and reliable.
- However insecure. Challenge-response behavior $r(c)$ of $m$-bit APUF:
  
  $$r(c) = [\Delta(c) \geq 0], \quad \Delta(c) = w^\top p,$$

  $w$: $(m + 1)$-d vector that is only a function of the signal delays in the APUF instance,
  $p$ (parity): $(m + 1)$-d vector that is only a function of the challenge $c$.

- Linear classification model; easy to learn given reasonable number of challenge-response pairs (CRPs).
Arbiter-based PUFs: XOR APUFs

**k-XOR APUF:**
- XOR-sum of $k$ APUF instances (sharing the challenge).
- #CRPs required in learning is believed to be exponential in $k$.
- Empirically vulnerable to *reliability-based attacks*, where the *reliability* information (probability of getting the same response) is accessible besides merely the response.
**k-XOR APUF:**

- XOR-sum of $k$ APUF instances (sharing the challenge).
- #CRPs required in learning is believed to be exponential in $k$.
- Empirically vulnerable to *reliability-based attacks*, where the *reliability* information (probability of getting the same response) is accessible besides merely the response.
Arbiter-based PUFs: XOR APUFs

$k$-XOR APUF:

- XOR-sum of $k$ APUF instances (sharing the challenge).
- #CRPs required in learning is believed to be exponential in $k$.
- Empirically vulnerable to *reliability-based attacks*, where the reliability information (probability of getting the same response) is accessible besides merely the response.
(x, y)-iPUF:

- Use two XOR APUFs as follows:

\[ r(c) = r_2(c_1, r_1(c), c_2), \]

- \( r_1 \) and \( r_2 \): respectively an \( x \)-XOR APUF and a \( y \)-XOR APUF, \((c_1, c_2)\): some fixed split of \( c \).

- As secure as \((x/2 + y)\)-XOR APUF, while moreover resilient to reliability-based attacks.
**Arbiter-based PUFs: Interpose PUFs (iPUFs)**

\[(x, y)-iPUF:
- Use two XOR APUFs as follows:
  \[ r(c) = r_2(c_1, r_1(c), c_2), \]
  where 
  - \( r_1 \) and \( r_2 \) are respectively an \( x \)-XOR APUF and a \( y \)-XOR APUF,
  - \((c_1, c_2)\): some fixed split of \( c \).
- As secure as \((x/2 + y)\)-XOR APUF, while moreover resilient to reliability-based attacks.
Our Contribution: Shift PUFs

\[ \mathbf{c} = [c_1, \ldots, c_m] \]

\[ \mathbf{c}^{(\ell)} = [c_{\ell+1}, \ldots, c_m, c_1, \ldots, c_{\ell}] \to \text{APUF Instance} \to r \]

- Prepend to APUF a (w.l.o.g., left) \textit{circular shift} operation.
- Model of challenge-response behavior \( r(\mathbf{c}) \) becomes:

\[
r(\mathbf{c}) = [\Delta_{\text{Shift}}(\mathbf{c}) \geq 0], \quad \Delta_{\text{Shift}}(\mathbf{c}) = \Delta(\mathbf{c}^{(\ell)}),
\]

\( \mathbf{c}^{(\ell)} \): the circular shift of \( \mathbf{c} \) by \( \ell \) bits.
Shift Displacement $\ell$ in Shift PUFs

- If $\ell$ is known by attacker, then the attacker could easily preprocess out the effect of the circular shift.
- Recall: securely generating some secret $\ell$ is exactly among the applications of PUFs.
- Could use *PUF-based key generation* with the underlying APUF instance to securely generate $\ell$;
- $\ell$ is only log $m$ bits long; will not harm the efficiency badly.

$$c = [c_1, \ldots, c_m] \quad \ell \quad \text{PUF-based Key Gen}$$

$$c^{(\ell)} = [c_{\ell+1}, \ldots, c_m, c_1, \ldots, c_\ell] \quad \text{APUF Instance} \quad r$$
Shift Displacement $\ell$ in Shift PUFs

- If $\ell$ is known by attacker, then the attacker could easily preprocess out the effect of the circular shift.
- Recall: securely generating some secret $\ell$ is exactly among the applications of PUFs.
- Could use *PUF-based key generation* with the underlying APUF instance to securely generate $\ell$;
- $\ell$ is only $\log m$ bits long; will not harm the efficiency badly.

\[
\mathbf{c} = [c_1, \ldots, c_m] \quad \ell \quad \text{PUF-based Key Gen} \quad \mathbf{c}(\ell) = [c_{\ell+1}, \ldots, c_m, c_1, \ldots, c_\ell] \quad \text{APUF Instance} \quad r
\]
Shift Displacement $\ell$ in Shift PUFs

- If $\ell$ is known by attacker, then the attacker could easily preprocess out the effect of the circular shift.
- Recall: securely generating some secret $\ell$ is exactly among the applications of PUFs.
- Could use *PUF-based key generation* with the underlying APUF instance to securely generate $\ell$;
  - $\ell$ is only $\log m$ bits long; will not harm the efficiency badly.

\[ c = [c_1, \ldots, c_m] \]
\[ \ell \rightarrow \text{PUF-based Key Gen} \]
\[ c^{(\ell)} = [c_{\ell+1}, \ldots, c_m, c_1, \ldots, c_\ell] \rightarrow \text{APUF Instance} \rightarrow r \]
Shift Displacement $\ell$ in Shift PUFs

- If $\ell$ is known by attacker, then the attacker could easily preprocess out the effect of the circular shift.
- Recall: securely generating some secret $\ell$ is exactly among the applications of PUFs.
- Could use PUF-based key generation with the underlying APUF instance to securely generate $\ell$;
- $\ell$ is only $\log m$ bits long; will not harm the efficiency badly.

$$c = [c_1, \ldots, c_m] \quad \ell \quad \text{PUF-based Key Gen}$$

$$c^{(\ell)} = [c_{\ell+1}, \ldots, c_m, c_1, \ldots, c_{\ell}] \quad \text{APUF Instance} \quad r$$
Shift Displacement $\ell$ in Shift PUFs

- If $\ell$ is known by attacker, then the attacker could easily preprocess out the effect of the circular shift.
- Recall: securely generating some secret $\ell$ is exactly among the applications of PUFs.
- Could use *PUF-based key generation* with the underlying APUF instance to securely generate $\ell$;
- $\ell$ is only $\log m$ bits long; will not harm the efficiency badly.

\[
c = [c_1, \ldots, c_m]
\]
\[
c^{(\ell)} = [c_{\ell+1}, \ldots, c_m, c_1, \ldots, c]\]
Security of Shift PUFs

- Linear classification model w.r.t. \( p^{(\ell)} \), the parity of \( c^{(\ell)} \), instead of \( p \).
- However \( \ell \) is unknown to the attacker; cannot apply linear methods.
- Eliminate the effect of \( \ell \) by enumerating all \( \ell = 0, 1, \ldots, m - 1 \).
- Natural and general approach: a \( \Theta(m^2) \)-d linear classification model.

\[
\begin{bmatrix}
  p_1 & p_1 & p_1 & \cdots & p_1 & p_1 \\
p_2 & p_1p_2p_3 & p_1p_3p_4 & \cdots & p_1p_{m-1}p_m & p_1p_mp_1 \\
p_3 & p_1p_2p_4 & p_1p_3p_5 & \cdots & p_m-1p_1 & p_mp_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
p_{m-1} & p_1p_2p_m & p_3p_1 & \cdots & p_{m-1}p_{m-3} & p_mp_{m-2} \\
p_m & p_2p_1 & p_3p_2 & \cdots & p_{m-1}p_{m-2} & p_mp_{m-1} \\
1 & 1 & 1 & \cdots & 1 & 1 \\
\end{bmatrix}
\]

- Conjecture: the attacker cannot do better than this approach.
Security of Shift PUFs

- Linear classification model w.r.t. $p^{(\ell)}$, the parity of $c^{(\ell)}$, instead of $p$.
- However $\ell$ is unknown to the attacker; cannot apply linear methods.
- Eliminate the effect of $\ell$ by enumerating all $\ell = 0, 1, \ldots, m - 1$.
- Natural and general approach: a $\Theta(m^2)$-d linear classification model.

$$
\begin{bmatrix}
  p_1 & p_1 & p_1 & \cdots & p_1 & p_1 \\
  p_2 & p_1p_2 & p_1 & \cdots & p_1p_{m-1} & p_m \\
  p_3 & p_1p_2 & p_1 & \cdots & p_{m-1} & p_m \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
  p_{m-1} & p_1p_2 & p_3 & \cdots & p_{m-1} & p_m \\
  p_m & p_2 & p_3 & \cdots & p_{m-1} & p_m \\
  1 & 1 & 1 & \cdots & 1 & 1
\end{bmatrix}
$$

- Conjecture: the attacker cannot do better than this approach.
Security of Shift PUFs

- Linear classification model w.r.t. $p^{(\ell)}$, the parity of $c^{(\ell)}$, instead of $p$.
- However $\ell$ is unknown to the attacker; cannot apply linear methods.
- Eliminate the effect of $\ell$ by enumerating all $\ell = 0, 1, \ldots, m - 1$.
- Natural and general approach: a $\Theta(m^2)$-d linear classification model.

\[
\begin{bmatrix}
  p_1 & p_1 & p_1 & \cdots & p_1 & p_1 \\
  p_2 & p_1 p_2 p_3 & p_1 p_3 p_4 & \cdots & p_1 p_{m-1} p_m & p_1 p_m p_1 \\
  p_3 & p_1 p_2 p_4 & p_1 p_3 p_5 & \cdots & p_m p_{m-1} p_1 & p_m p_1 p_2 \\
  \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
  p_{m-1} & p_1 p_2 p_{m-1} & p_3 p_1 & \cdots & p_{m-1} p_{m-3} p_{m-1} & p_{m-1} p_{m-2} p_{m-1} \\
  p_m & p_2 p_1 & p_3 p_2 & \cdots & p_{m-1} p_{m-2} & p_m p_{m-1} p_{m-1} \\
  1 & p_2 & 1 & \cdots & 1 & 1 \\
  \end{bmatrix}
\]

- Conjecture: the attacker cannot do better than this approach.
“Squaring” Arbiter-based PUFs

By substituting APUFs with shift PUFs, all arbiter-based PUF designs might benefit from the $\Theta(m^2)$ enhancement.

- Any machine learning complexity $T(m)$ becomes $T(m^2)$.\(^2\)
- Remark: not for turning insecure design into secure design.
- Substantially adding difficulties to the attacks.
- Alternatively, reducing time and/or circuit complexities of the hardware while preserving the machine learning complexity.

\(^2\)Strictly speaking it is $\Theta(T(m^2))$, as long as $T$ is polynomial in $m$; also note that $\Theta(T(m^2)) = \Theta(T(m)^2)$ in such case.
“Squaring” Arbiter-based PUFs

By substituting APUFs with shift PUFs, all arbiter-based PUF designs might benefit from the $\Theta(m^2)$ enhancement.

- Any machine learning complexity $T(m)$ becomes $T(m^2)$.$^2$
- Remark: not for turning insecure design into secure design.
- Substantially adding difficulties to the attacks.
- Alternatively, reducing time and/or circuit complexities of the hardware while preserving the machine learning complexity.

$^2$Strictly speaking it is $\Theta(T(m^2))$, as long as $T$ is polynomial in $m$; also note that $\Theta(T(m^2)) = \Theta(T(m)^2)$ in such case.
“Squaring” Arbiter-based PUFs

By substituting APUFs with shift PUFs, all arbiter-based PUF designs might benefit from the $\Theta(m^2)$ enhancement.

- Any machine learning complexity $T(m)$ becomes $T(m^2)$.
- Remark: not for turning insecure design into secure design.
- Substantially adding difficulties to the attacks.
- Alternatively, reducing time and/or circuit complexities of the hardware while preserving the machine learning complexity.

---

$^2$Strictly speaking it is $\Theta(T(m^2))$, as long as $T$ is polynomial in $m$; also note that $\Theta(T(m^2)) = \Theta(T(m)^2)$ in such case.
“Squaring” Arbiter-based PUFs

By substituting APUFs with shift PUFs, all arbiter-based PUF designs might benefit from the $\Theta(m^2)$ enhancement.

- Any machine learning complexity $T(m)$ becomes $T(m^2)$.

2Remark: not for turning insecure design into secure design.

- Substantially adding difficulties to the attacks.

- Alternatively, reducing time and/or circuit complexities of the hardware while preserving the machine learning complexity.

---

2Strictly speaking it is $\Theta(T(m^2))$, as long as $T$ is polynomial in $m$; also note that $\Theta(T(m^2)) = \Theta(T(m)^2)$ in such case.
“Squaring” Arbiter-based PUFs

By substituting APUFs with shift PUFs, all arbiter-based PUF designs might benefit from the $\Theta(m^2)$ enhancement.

- Any machine learning complexity $T(m)$ becomes $T(m^2)$.\(^2\)
- Remark: not for turning insecure design into secure design.
- Substantially adding difficulties to the attacks.
- Alternatively, reducing time and/or circuit complexities of the hardware while preserving the machine learning complexity.

\(^2\text{Strictly speaking it is } \Theta(T(m^2)), \text{ as long as } T \text{ is polynomial in } m; \text{ also note that } \Theta(T(m^2)) = \Theta(T(m)^2) \text{ in such case.}\)
To empirically verify the conjecture: $T(m)$ becomes $T(m^2)$.

- Various kinds of arbiter-based PUFs.
- Various commonly used attacks.
Thank you.