A Design Methodology for Stealthy Parametric Trojans and Its Application to Bug Attacks

CHES 2016 - Santa Barbara, CA

Samaneh Ghandali[1], Georg T. Becker[2], Daniel Holcomb[1], and Christof Paar[1,2]

[1] University of Massachusetts Amherst, Amherst, USA

[2] Horst Görtz Institut for IT-Security, Ruhr-Universität Bochum, Bochum, Germany

August 19, 2016

Motivation: Hardware Trojans

- Many potential attack vectors
  - Malicious foundry/company
  - Malicious employee
  - 3rd party IP cores
  - Government request
  - Hacker attacks
  - …

Criminals

State actors

Hacktivists

Motivation

- Small changes at certain points can break/weaken crypto
  - RNG [Becker et al, CHES 2013]
  - Bug Attack [Biham et al, CRYPTO 2008 and Journal of Cryptology 2015]

- So, why do we trust our chips to function as intended?
  - They seem to do right thing for any practical # of operations [[FPDIV, Edelman, SIMA 1997].
  - The logic gates appear correct.

- In this work, we will show that even if logic gates appear correct, and circuit usually does right things, Trojans can still exist.

Outline

- Introduction
- Path delay fault (PDF)
- Creating a stealthy PDF
  - Phase I: Path Selection
  - Phase II: Delay Distribution along path
- Bug Attack on ECDH
- Conclusion

Why Design Trojans?

- Trojan detection and design are closely related
  - To design effective detection mechanisms, we need an understanding of how Hardware Trojans can be built.

- We examine how particularly stealthy parametric Trojans can be introduced to a target circuit.

Motivating Example: Bug Attacks

- Setting: RSA digital signature running on server or embedded device
- Secret key k leaks if there is ONE fault in multiplication during exponentiation m^k mod n.

[Biham, Carmeli, Shamir , CRYPTO 2008 & Journal of Cryptology 2015]

message m

$Sig' = m^k \bmod n$

Sig'

faulty sig' leaks k via Chinese reminder theorem

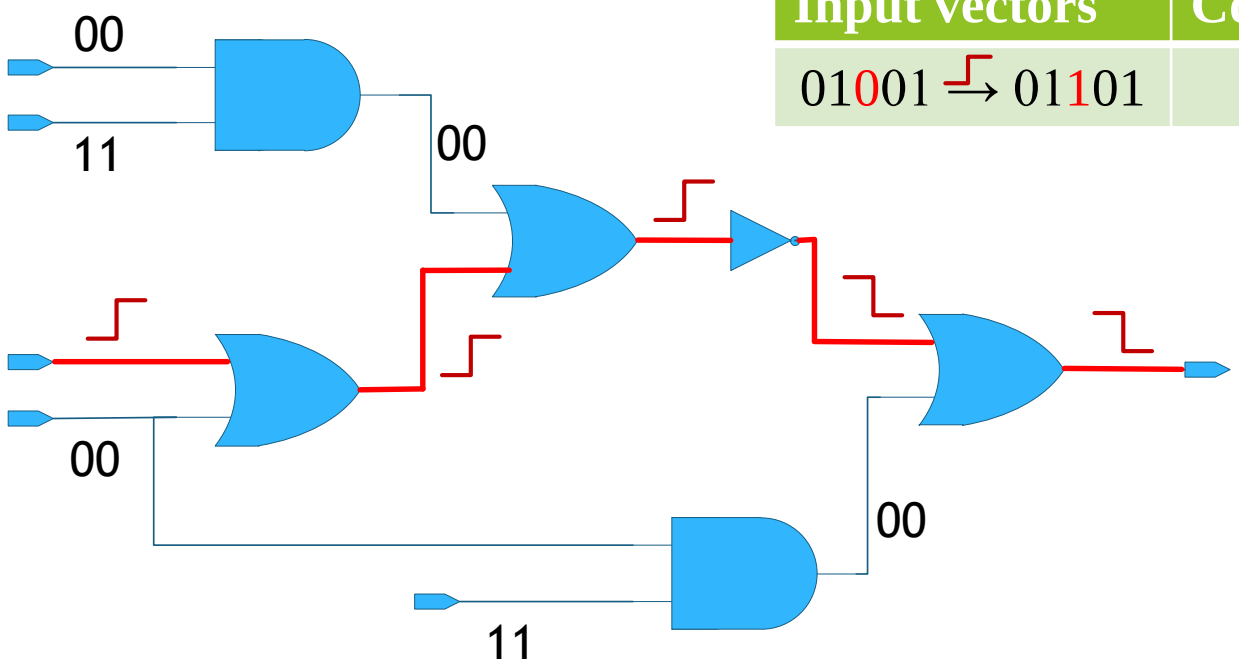
- Research Challenge: Low-level manipulation of integer multiplier such that

A * B
- = C almost all inputs
- ≠ C a **few** poisonous inputs A', B'

Path Delay Fault

- Stuck-at fault model: easily detected and not rare
- Path delay fault model:

Input vectors	Correct output	Faulty output
01001 ⇉ 01101	1⇉ 0	1⇉1



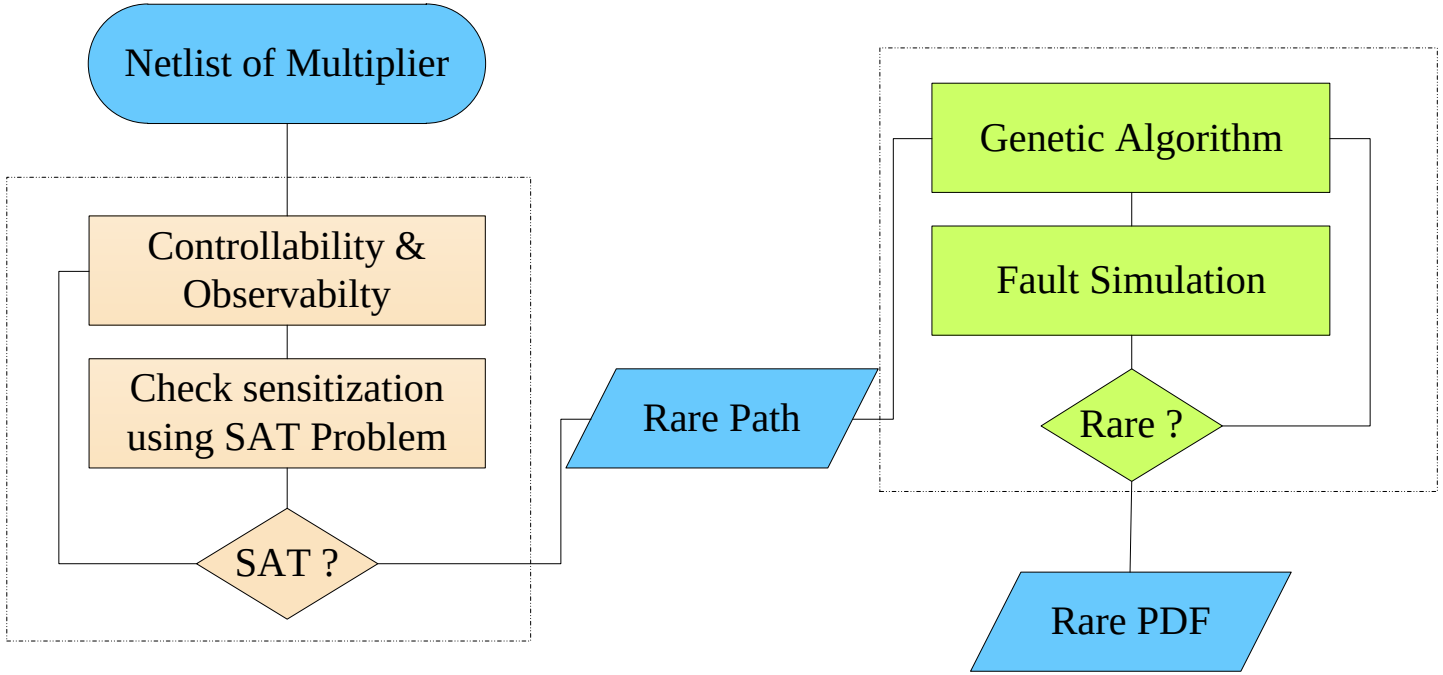
Delay-based Trojan

- Two properties for a viable delay-based Trojan:
  - Triggerability: For secret inputs, which are known to the attacker, cause an error with certainty or relatively high probability.
  - Stealthiness: For randomly chosen inputs, cause an error with extremely low probability.

Outline

- Introduction
- Path delay fault (PDF)
- Creating a stealthy PDF
  - Phase I: Path Selection
  - Phase II: Delay Distribution along path
- Bug Attack on ECDH
- Conclusion

Proposed method for creating a stealthy PDF

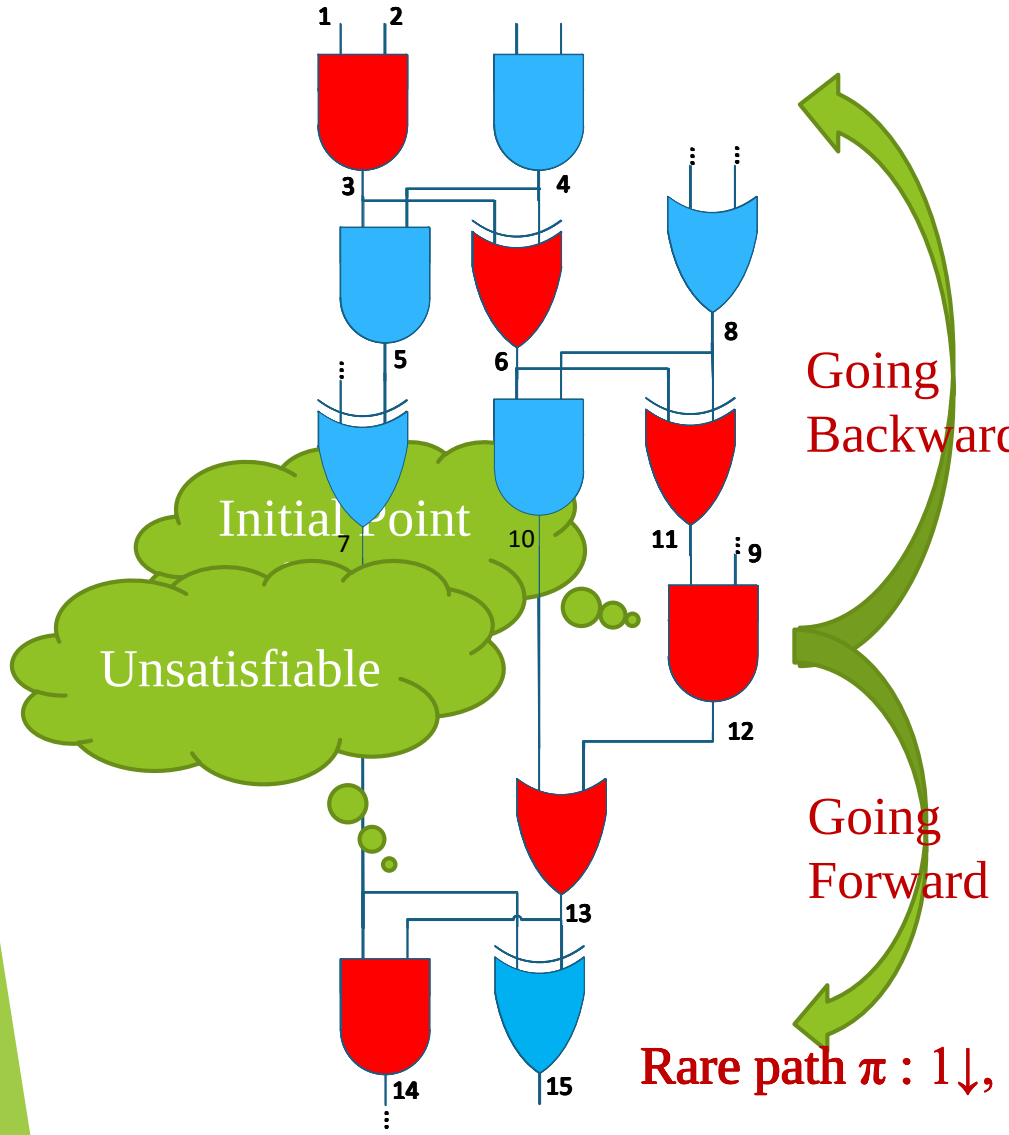


- Phase I: Path Selection
  - Finding a rarely sensitized path
  - Selection guided by Controllability and Observability metrics from testing
  - SAT-based check ensures path is sensitizable.
- Phase II: Delay Distribution
  - Decide where on rare path to add delay
  - Increase the delay of the rare path to occur a PDF
  - Stealthiness problem: Cause faults on intersecting paths if not assigning delay carefully.
  - Using Genetic Algorithm to choose delay of each gate of the rare path

Finding a rarely sensitized path

- Path π is seeded with a single hard to sensitize transition
- Extend π backward until reaching primary inputs
- Extend π forward until reaching primary outputs
- SAT-check ensures that π remains sensitizable each time π is extended.
  - Also produces vector pair (Poison Inputs) that can be used to trigger the fault

Path Selection Example: 3-bit Wallace Tree Multiplier



Node	Cont-0	Cont-1	Obs
1	0.5	0.5	0.37
2	0.5	0.5	0.42
3	0.75	0.25	0.61
4	0.75	0.25	0.61
5	0.93	0.07	0.67
6	0.63	0.37	0.81
7	0.81	0.19	0.89
8	0.84	0.16	0.61
9	0.97	0.03	0.66
10	0.91	0.09	0.79
11	0.66	0.34	0.97
12	0.98	0.02	0.74
13	0.89	0.11	0.81
14	0.97	0.03	0.94
15	0.77	0.23	1

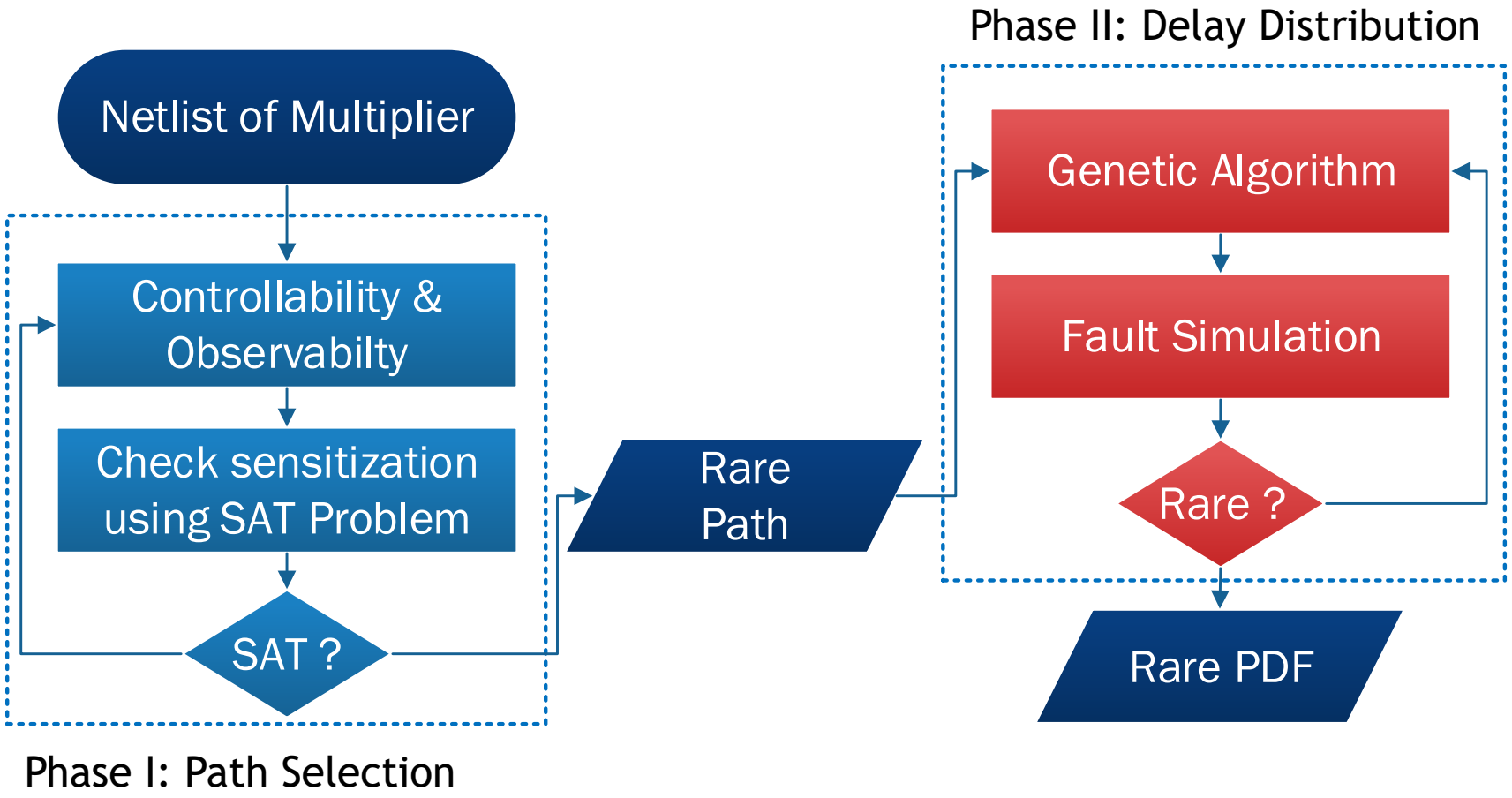
Rare path π : 1↓, 3↓, 6↓, 11↓, 12↓, 13↓, 14↓

Evaluation of Path Selection

Fault simulation of rare path and 750 random paths of 32-bit Wallace tree multiplier.

Path selection algorithm finds a path that is much rare than the random search.

Proposed method for creating a stealthy PDF

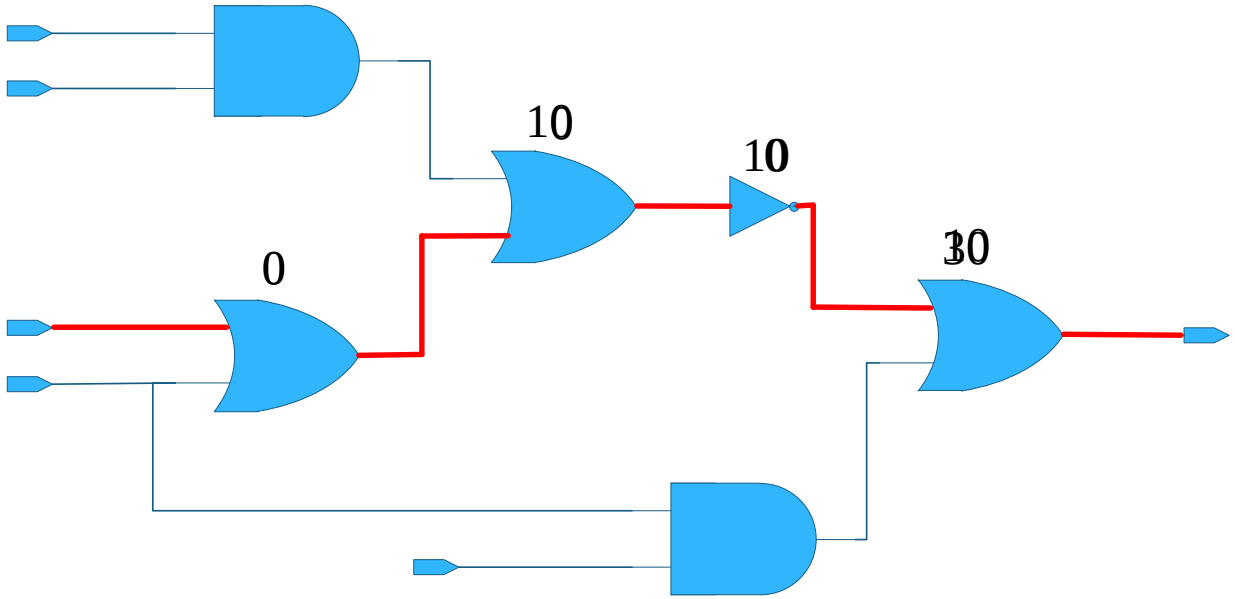


Delay Distribution Along Chosen Path

- Delay of the path is increased so that it will exceed the clock period and an error will occur when the path is sensitized
- Where on path to increase delay?
- Stealthiness problem: additional delay can cause errors on paths that intersect or overlap with chosen path
  - We want to minimize it
- Genetic Algorithm is used to decide the delay of each gate to cause a PDF which is triggered by poison inputs but rarely triggered by other inputs
  - Fitness function of GA is empirical probability from simulation of causing an error when random input vectors are applied to the circuit

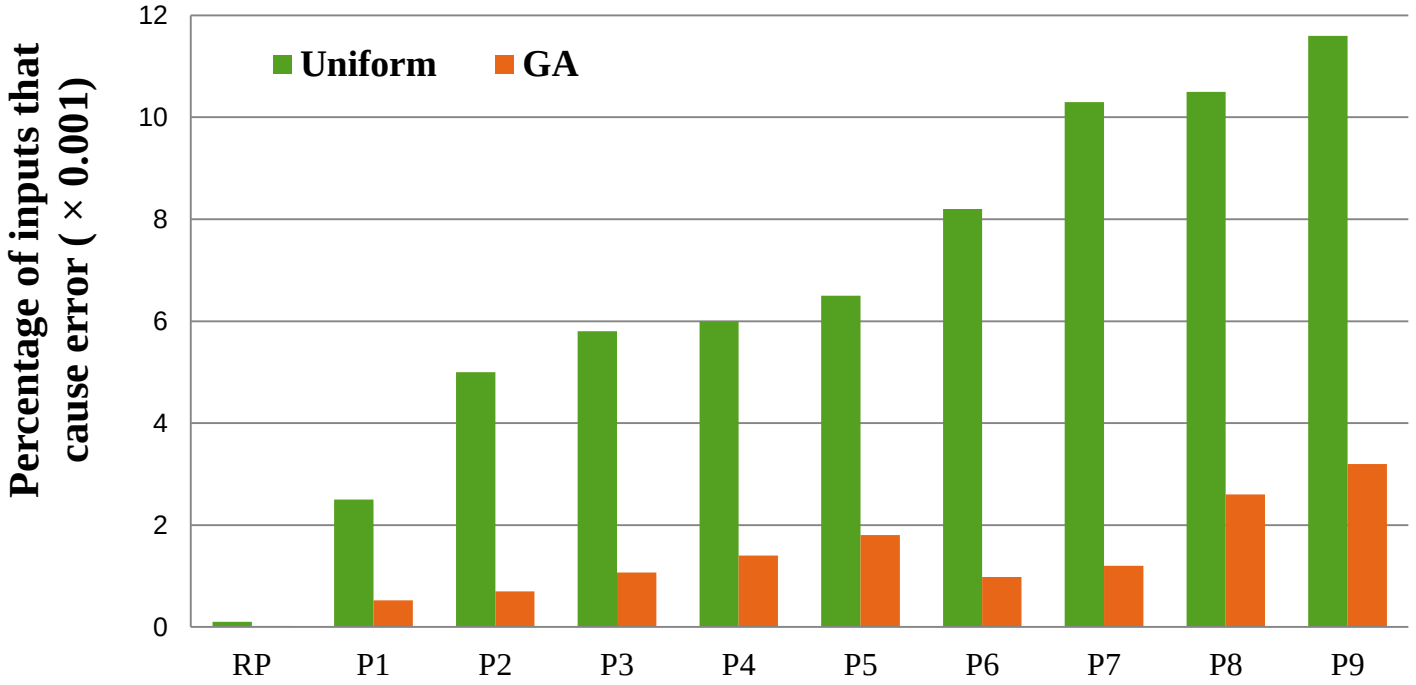
Delay Distribution Example:

- Distributing a delay of 30 units on the selected path:
  - Without GA: the Probability of detection is 0.22
  - Using GA: the Probability of detection is 0.16



Evaluation of Delay Distribution

- Error probability of circuit before and after optimizing delay assignment of rare path and 9 other best ones in a 32 × 32 Wallace tree multiplier.



Path delay = 3276 ps

Clock Period = 2530 ps

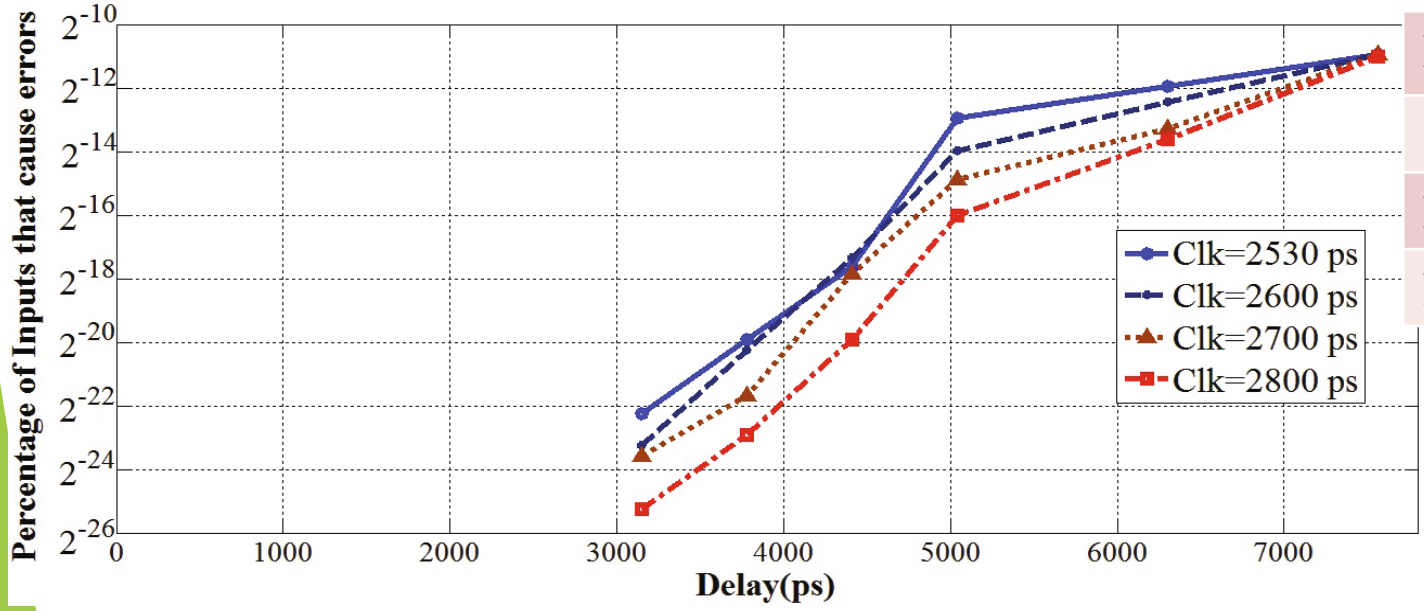
- GA reduces number of faults, while not affecting triggerability.

Overall Evaluation

- Nominal critical path is 2520 ps. If rare path gets delay 2530ps, how often does circuit delay exceed 2520ps?

	Probability
Uniform Delay Distribution	0.0003 (57/200k sim. vectors)
Genetic Algorithm	$<2^{-26}$ (0/260M sim. vectors)

- Clock period usually significantly longer than critical delay
- Even when path delay far exceeds nominal critical path, errors are still very rare.



Nominal critical path	2520 ps
Clock period	2800 ps
Delay of rare path after modification	3150 ps
Probability of output error	$2^{-25.25}$

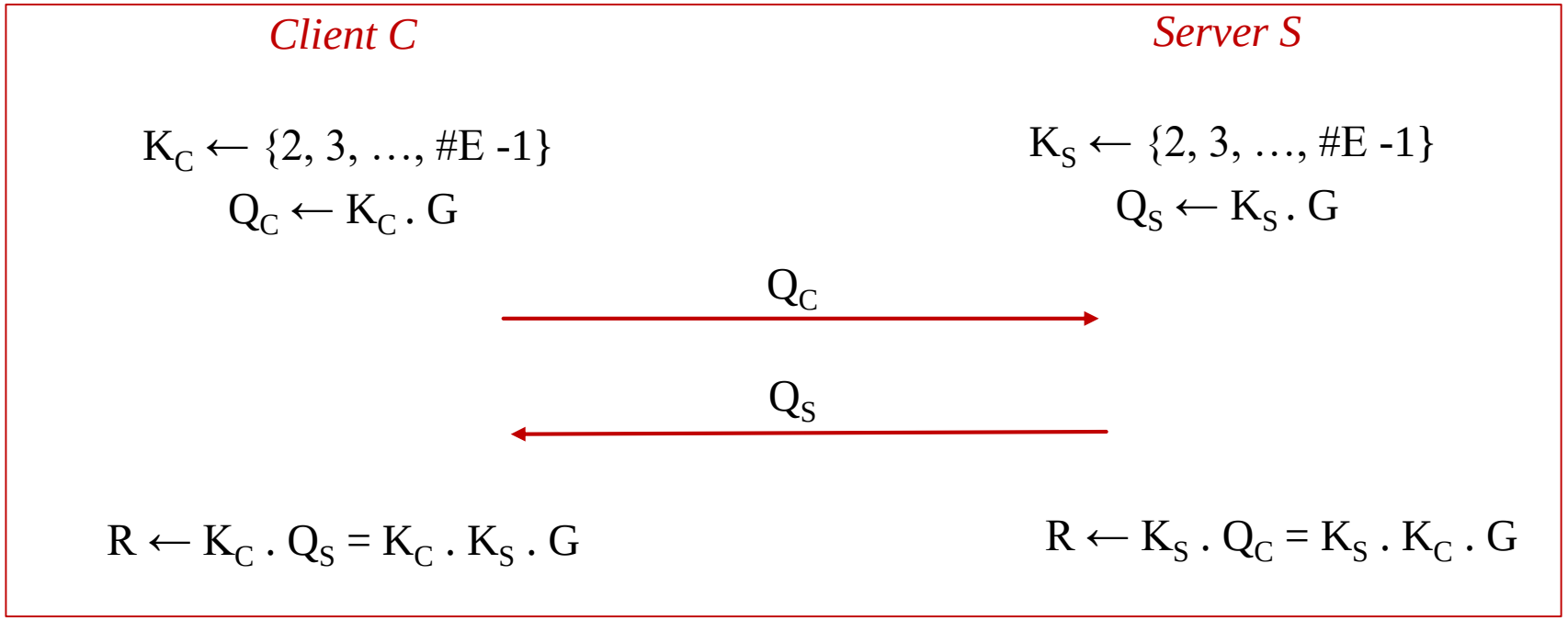
- Now we have a stealthy triggerable fault for bug attack

Outline

- Introduction
- Path delay fault (PDF)
- Creating a stealthy PDF
  - Phase I: Path Selection
  - Phase II: Delay Distribution along path
- Bug Attack on ECDH
- Conclusion

ECDH Algorithm

- Elliptic Curve Diffie-Hellman Key Exchange (ECDH) between *C* and *S*



Bug Attack on ECDH with Montgomery Ladder

- Main idea:
  - Send a poisonous point Q_c so that an error occurs if the most significant unknown key bit is 1 (or 0)

  ⇒ Attacker learns one key-bit per message

- Problem:
  - Handshake needs to be completed to detect an error

  ⇒ Point Q_c cannot be arbitrarily chosen

  ⇒ Therefore attack complexity very high to search for a fitting Q_c

- Solution:
  - We specifically target Montgomery Ladder step
  - Introduce a one-time only **pre-computation step** to find a set of "good" points

  ⇒ Attack much more reasonable since the computation complex part only needs to be done once per curve parameters and Trojan

Attack complexity

Target: 256-bit ECDH with Montgomery Ladder scalar multiplication

Failure probability (of one multiplication)	**2^{-64}**	**2^{-48}**	**2^{-32}**
Precomputation complexity (point additions)	2^{67}	2^{51}	2^{35}
Storage requirements	14 PB	55 TB	215 GB
Attack complexity (Montgomery Ladder steps)	2^{47}	2^{39}	2^{31}

Conclusion

- Introducing a new type of parametric hardware Trojans based on rarely-sensitized path delay faults.
- Presenting a SAT-based algorithm which searches the circuit for paths that are extremely rarely sensitized.
- Presenting a delay distribution method using Genetic Algorithm
- Modifying a 32-bit multiplier so that for extremely rare inputs faulty responses are computed.
- Bug attack against ECDH implementation.

Thank You!

Attack complexity

Target: 256-bit ECDH with Montgomery Ladder scalar multiplication

Failure probability (of one multiplication)	**2^{-64}**	**2^{-48}**	**2^{-32}**
Precomputation complexity (point additions)	2^{67}	2^{51}	2^{35}
Storage requirements	14 PB	55 TB	215 GB
Attack complexity (Montgomery Ladder steps)	2^{47}	2^{39}	2^{31}

Sasdrich et al. [22] can compute roughly 2^40 Montgomery ladder steps per second on a Zynq 7020 FPGA (curve 25519

⇒Computation complexity in the seconds for the attack

⇒Precomputation possible for well-funded adversaries