Parallel Synchronous Code Generation for Second Round Light Weight Candidates

Pantea Kiaei, Archanaa S. Krishnan, Patrick Schaumont

Worcester Polytechnic Institute

Virginia Tech

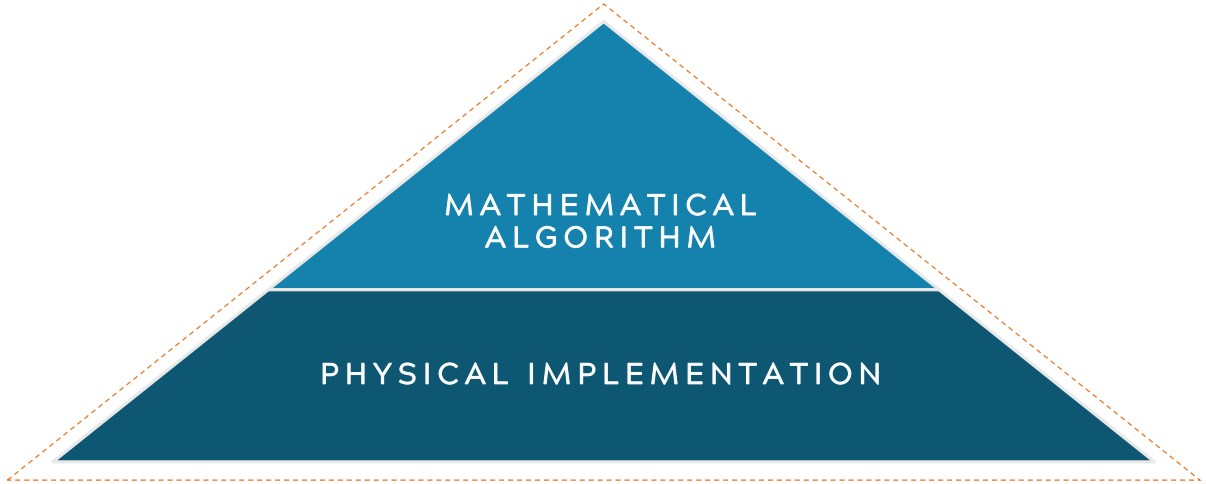
Outline

1. Background
2. Timing side-channel
3. Timing aspects of AEAD ciphers
4. Predictable-time solutions
5. Parallel synchronous programming
6. Results

Background

- AEAD algorithm
  - Confidentiality
  - Integrity
  - Authenticity
- Implementation
  - Side-channel analysis
  - Fault attacks

- Cryptanalysis on the algorithm



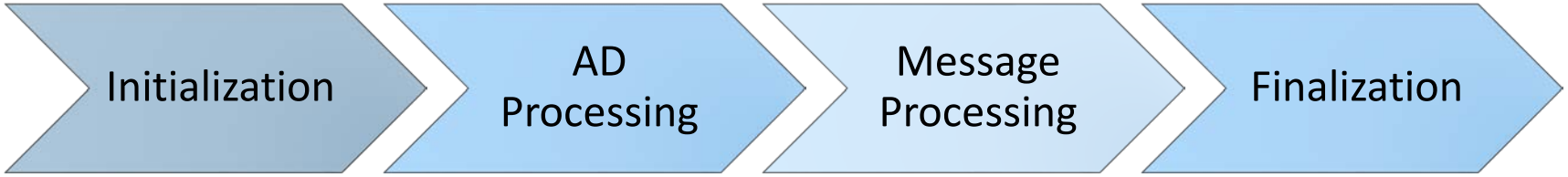
  - Timing analysis
  - Power analysis
  - Fault injection
  - …

Timing attacks

- Sources of non-constant run-time:
  - Algorithm
    - Data-dependent run-time
  - Memory hierarchy
    - Cache-hit *vs.* cache-miss
  - Shared resources
- Timing side-channel
  - Algorithm
    - Table lookups
  - Memory hierarchy
    - Last Level Cache (LLC): Flush+Reload [1], Prime+Probe [2]

[1] Yuval Yarom, Katrina Falkner. “FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack”. USENIX Security 2014
[2] Dag Arne Osvik, Adi Shamir, Eran Tromer. “Cache attacks and countermeasures: the case of AES”. Cryptographers' track at the RSA conference 2006

AEAD ciphers

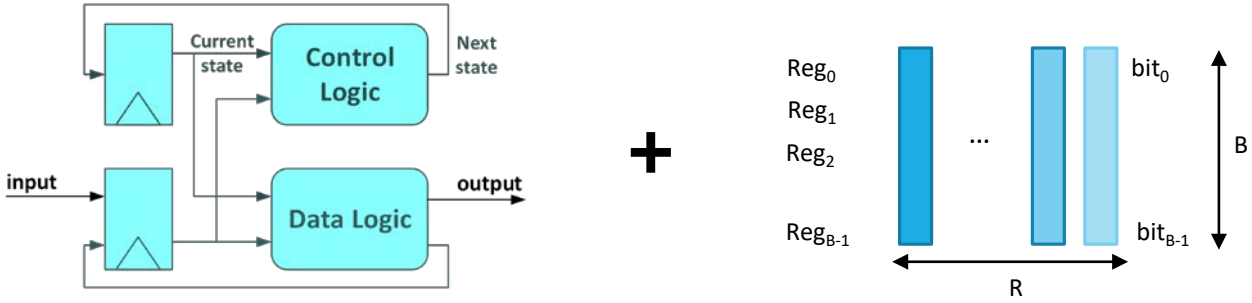


- Timing side-channel
  - Recognize different phases
    - Facilitate power SCA and fault injection
  - Gain information about the internal state

Predictable-time solutions

- Bitslicing
  - Every thing computed
    - no table lookups
  - Data scattered over many locations
    - no memory hierarchy timing leakage
  - Normal control logic

- Parallel Synchronous Programming [3]
  - Implements FSMD in software
  - Adds the control logic into the bitslicing
    - no algorithm-related unpredictability



[3] P. Kiaei and P. Schaumont. “Synthesis of Parallel Synchronous Software”. IEEE Embedded Systems Letters 2020

Parallel Synchronous Programming (PSP)

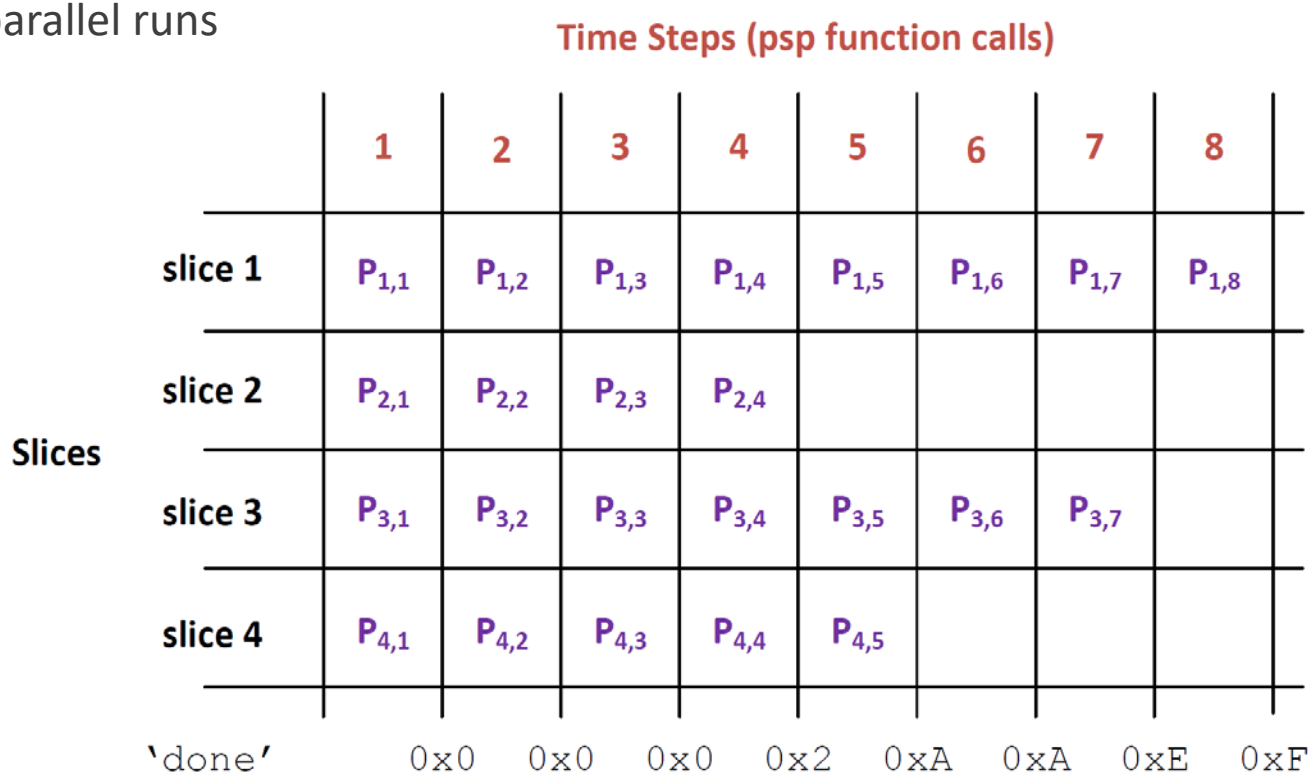
- PSP kernel
  - Generated by automated tool (PSPCG)
  - Equivalent to one clock cycle of FSMD
- PSP wrapper
  - Calls the PSP kernel

```
void kernel(..., MDTYPE* done) {
...
NOT1(rst, n01_);
XOR2(fsmd_reg[0], CNT[0], n02_);
XOR2(CNT[1], fsmd_reg[1], n03_);
OR2(n02_, n03_, n04_);
...
DFF(clk, n00_[0], fsmd_reg[0]);
DFF(clk, n00_[1], fsmd_reg[1]);
DFF(clk, n00_[2], fsmd_reg[2]);
DFF(clk, n00_[3], fsmd_reg[3]);
}
```

```
int main() {
// prepare inputs in bitsliced format
...
// reset:
...
// keep calling the kernel until all
// calculations complete:
while (done != 0xffffffff) {
     kernel(..., &done);
}
return 0;
}
```

PSP implementation of LWC candidates

- Predictable run-time
  - The longest calculation among the parallel runs
- Indistinguishable AEAD phases



Implementations

- Implemented PSP version of a few second round LWC candidates
  - *Not a comparison among the candidates*
- Compared the PSP version and normal implementation provided by submissions
- Compared the PSP version and bitsliced implementation generated by the Usuba compiler

Results: PSP *vs.* reference implementation

AEAD	Implementation	Code size(B)	Cycles/byte
Ascon-128	Reference	15,709	2,751
	PSP	210k	6,855
WAGE-$\mathcal{AE}$-128	Reference	24,439	9,305
	PSP	47,304	21,032

Table 2: Overhead of complete PSP implementations of Ascon-128 and WAGE-$\mathcal{AE}$-128 when compared with reference implementation. The measurements were computed for processing 8B of plaintext and 8B of associated data

Results: PSP *vs.* bitsliced – code size

Cipher/permutation	Implementation	Code size (B)	Cycle count (cycles)
ACE permutation	Reference	14,937	261,833
	Usuba	115k	1,408,096
ACE-$\mathcal{AE}$-128	PSP	84,280	34,662

Table 3: Code size and cycle count comparison of ACE-$\mathcal{AE}$-128 PSP core implementation with ACE permutation reference implementation and its Usuba bitsliced implementation.

Results: PSP *vs.* bitsliced – logic

Cipher	Code Generator	AND	ORR	EOR	MVN	total
Ascon-p^{12}	Usubac	3840	0	16128	4657	24625
	PSPCG	6381	6034	1669	1324	15408
GIFT-128	Usubac	3840	1280	14080	1360	20560
	PSPCG	756	418	262	137	1573

Table 4: Comparison of number of instructions resulting from Usubac and PSPCG

Results: PSP *vs.* bitsliced – memory spill

Cipher	AND	ORR	EOR	MVN	MOV	LDR	STR	overhead
ACE permutation	64	0	1168	832	11229	1469	25	86.04%
GIFT-128	192	64	704	66	6725	2041	18	89.54%

Table 5: Register spill of the bitsliced code generated by Usubac

Cipher	AND	ORR	BIC	EOR	ORN	MVN	MOV	LDR	STR	overhead
Ascon-p^{12}	1732	1296	281	808	1265	123	4904	10277	3862	77.57%
GIFT-128	530	57	30	58	54	2	120	1415	971	77.42%

Table 6: Register spill of the PSP code generated by PSPCG

Conclusion

- Naïve implementations of AEAD ciphers can result in exploitable timing side-channel
- Parallel synchronous programming can prevent timing side-channel of AEAD ciphers
- Parallel synchronous programs in contrast to bitslicing include the control logic therefore are more secure while also being more compact

Thank you

https://github.com/Secure-Embedded-Systems/psp-nistlwc20