サイドチャネル攻撃耐性の効率的な評価が可能なフレームワーク

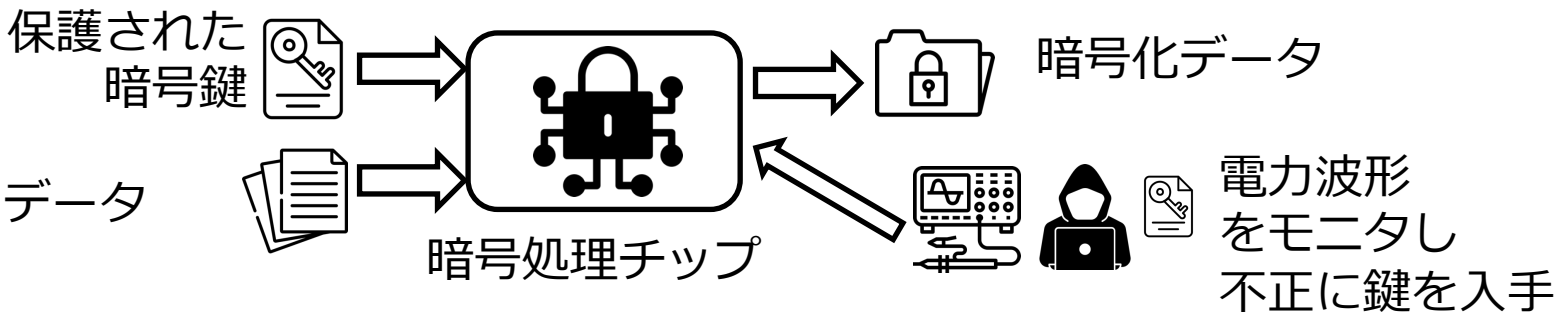
～高位合成によるAES回路の評価をケーススタディとして ～

小島　拓也
東京大学

ハードウェアセキュリティの重要性

- 信頼の基点「Root of Trust」はハードウェアがベースとなるのが一般的
- IoTデバイスは攻撃者からの物理的な攻撃の脅威

- **サイドチャネル攻撃**
  - 消費電力や電磁波、処理時間などのサイドチャネル情報をもとにICチップなどの内部状態を不正に取得する攻撃
  - 近年は機械学習の利用により解析性能が向上、脅威度が高まっている



ハードウェアセキュリティの重要性

- 信頼の基点「Root of Trust」はハードウェアがベースとなるのが一般的
- IoTデバイスは攻撃者からの物理的な攻撃の脅威

- **サイドチャネル攻撃**
  - 消費電力や電磁波、処理時間などのサイドチャネル情報をもとにICチップなどの内部状態を不正に取得する攻撃
  - 近年は機械学習の利用により解析性能が向上、脅威度が高まっている

課題: サイドチャネル攻撃耐性の評価にかかる時間的コスト

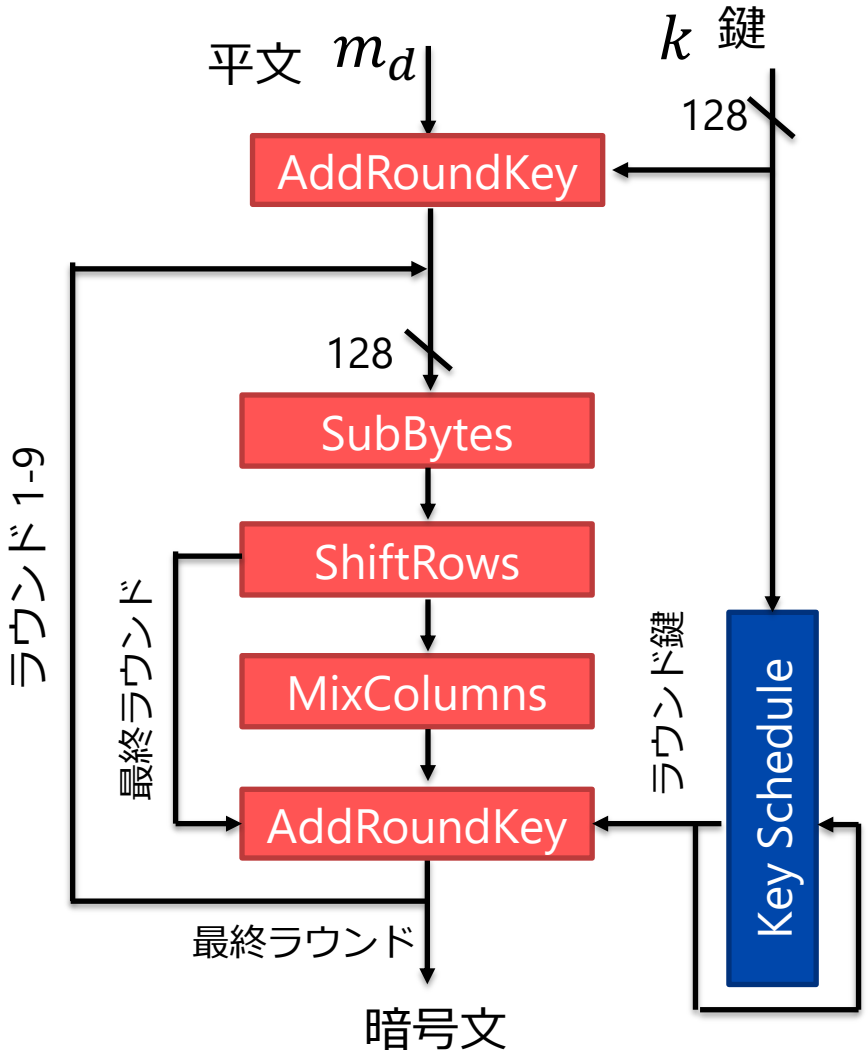
1. 消費電力波形の自動取得などの評価環境の構築
2. 膨大な測定データに対する解析時間の長さ

本発表の流れ

- 貢献の概略
  - 標準的に利用されるFPGA向けにAXIバスを持つ設計テンプレートによりサイドチャネル攻撃耐性の評価にかかる実装コストを削減
  - 時間のかかる相関係数解析をメニーコア、GPUを活用して高速化: 最大450倍
  - 既存のフレームワークとの互換性を重視してプラグインとして統合
- 流れ
  1. AESモジュールに対するサイドチャネル攻撃
  2. 関連研究: 既存フレームワークの紹介
  3. 新たなフレームワークの提案
     1. 相関係数解析の高速化実装
     2. FPGA用設計テンプレート
  4. 評価
     1. 相関係数解析の実行時間比較
     2. 実装方法の異なるAESモジュールに対する攻撃結果

AESモジュールに対するサイドチャネル攻撃

AESにおける暗号化の流れ



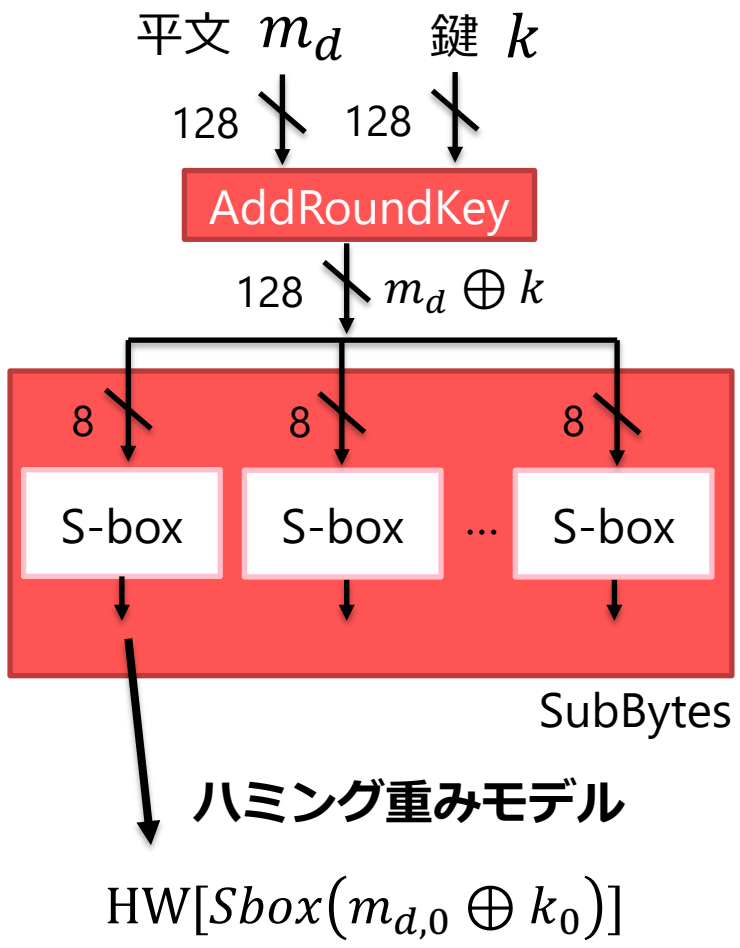
- Advanced Encryption Standard (AES)
  - 共通鍵暗号
  - ブロック暗号
    - 鍵長: 128bit, 192bit, 256bit
- 共通の処理(ラウンド)を複数回繰り返す
  - 128bit長の場合は10回
- 各ラウンドは4つの処理で構成
  - SubBytes: S-boxによる非線形変換
  - ShiftRows : 行シフト
  - MixColumns: 行列変換
  - AddRoundKey: ラウンド鍵とのXOR

相関係数解析による攻撃

- Correlation Power Analysis (CPA)
  - サイドチャネル攻撃で利用される手法の1つ
  - 多数の暗号化を実施して、入力した平文(+暗号文)と消費電力のペアを取得しその相関を解析して内部の鍵を推定する
  - ピアソンの相関係数ρが用いられる
    - X: 観測した消費電力
    - Y: 鍵候補から予測される電力値
    - 正解鍵の場合には高い相関が現れることを利用して鍵を推定

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$$

予測リーケージ: 消費電力のモデル化



- 暗号処理の中間データと消費電力に相関
  - ダイナミック電力は信号のスイッチングにより消費
- 電力モデルは単純なものを用いる
  - 相関がわかれば良いので正確に見積もる必要はない
  - ハミング重みモデル: 中間値のハミング重み
  - ハミング距離モデル: 中間値の変化前後のハミング距離
- 典型的な攻撃対象: S-Boxの出力値
  - 予測した部分鍵と平文から計算可能

相関係数解析による鍵の推定

トレースデータ (正解鍵 k=03)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
平文	08	80	c4	5c	83	db	4d	4a	4d	10	55	48	1d	2a	7d	1c
消費電力	10.91	12.00	11.76	11.66	12.35	11.01	12.20	12.34	11.90	12.15	11.46	11.76	10.97	11.70	12.36	10.30

全鍵候補$\hat{k}$について予測リーケージの計算　$\mathrm{HW}[Sbox](m_d \oplus \hat{k})$

鍵候補		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\hat{k}$ = 0	S-Box	30	cd	1c	4a	ec	b9	e3	d6	e3	ca	fc	52	a4	e5	ff	9c
	HW	2	5	3	3	5	5	5	5	5	4	6	3	3	5	8	4
…																	
$\hat{k}$ = ff																	

8トレースで相関係数を計算
トレース数不足により正しくない推定

鍵候補 $\hat{k}$	00	01	02	03	04		b5	
相関係数 ρ	0.58	0.20	-0.86	0.64	-0.51	…	**0.90**	…

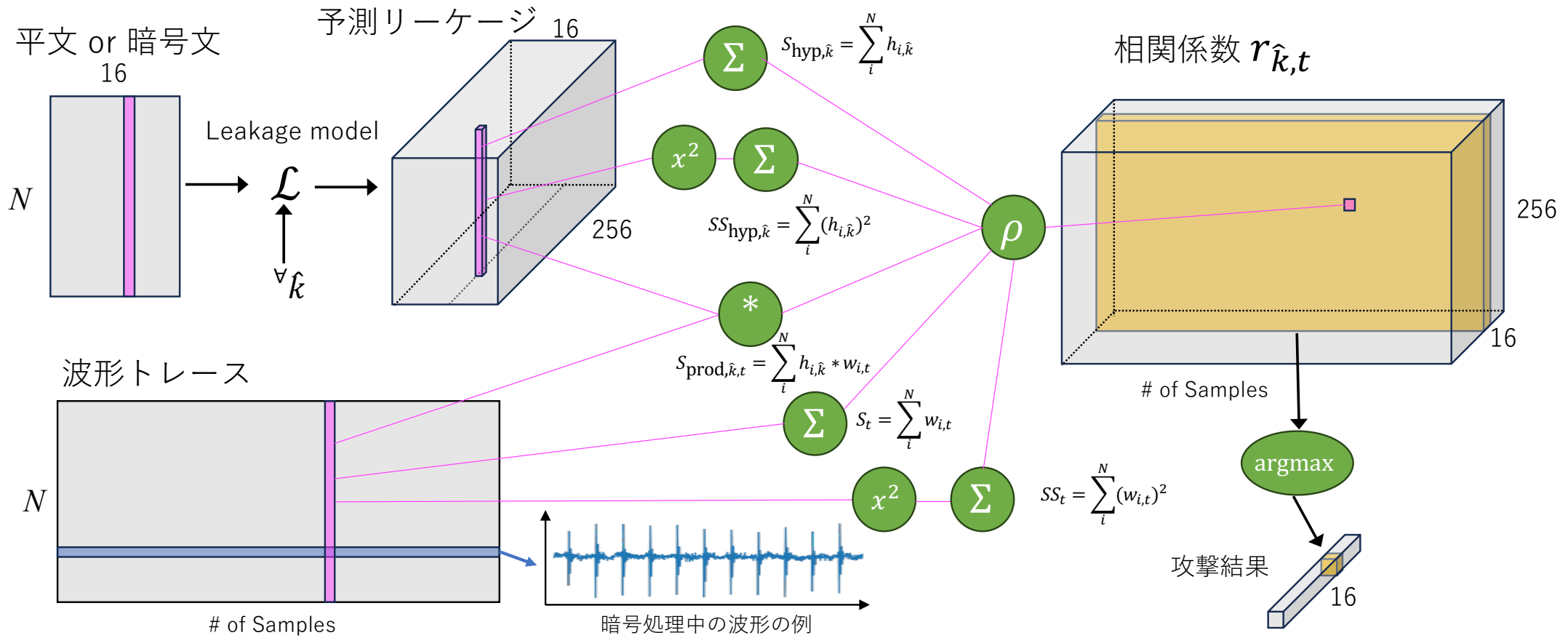
全16トレースで相関係数を計算
正しい鍵の推定に成功

鍵候補 $\hat{k}$	00	01	02	03	04	
相関係数 ρ	0.46	-0.05	-0.33	**0.82**	-0.07	…

実際の攻撃に必要な計算量

- 消費電力は複数のサンプルを持つ波形である
- 16の鍵バイトごとにそれぞれ計算

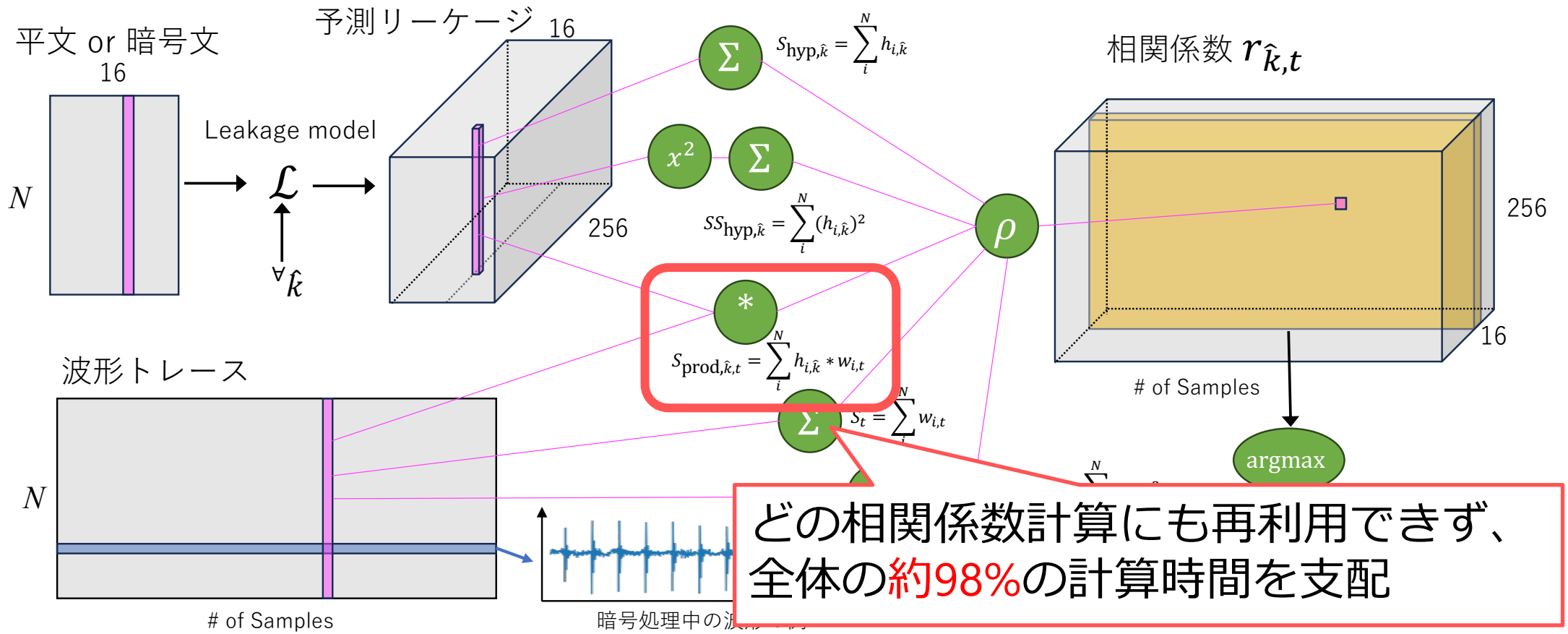
波形のサンプル数,トレース数が大きいと計算量が増大



実際の攻撃に必要な計算量

- 消費電力は複数のサンプルを持つ波形である
- 16の鍵バイトごとにそれぞれ計算

波形のサンプル数,トレース数が大きいと計算量が増大



どの相関係数計算にも再利用できず、全体の約98%の計算時間を支配

関連研究
既存フレームワークの紹介

サイドチャネル攻撃研究用フレームワーク

フレームワーク/文献	ソフトウェア			ハードウェア	
	予測モデル	実装	解析の高速化	ターゲット	トレース取得方法
RamDPA	S-Box出力	C++	マルチスレッド		
DareDevil	LUTファイル	C++	マルチスレッド		
T. Swamy, *et al.*	最終Round	C++	OpenCL/CUDA		
H. Gamaarachchi, *et al.*	最終Round		CUDA		
FOBOS3	別途計算	Python		FOBOS DUT	FOBOS Shiled
SASEBO/SAKURA	最終Round	C#	マルチスレッド	SASEBO/SAKURA シリーズ	VISA互換 オシロスコープ
ChipWhisperer	複数利用可能	Python+ Jupyter	n/a	CWシリーズ	ChipWhisperer-Nano, etc

サイドチャネル攻撃研究用フレームワーク

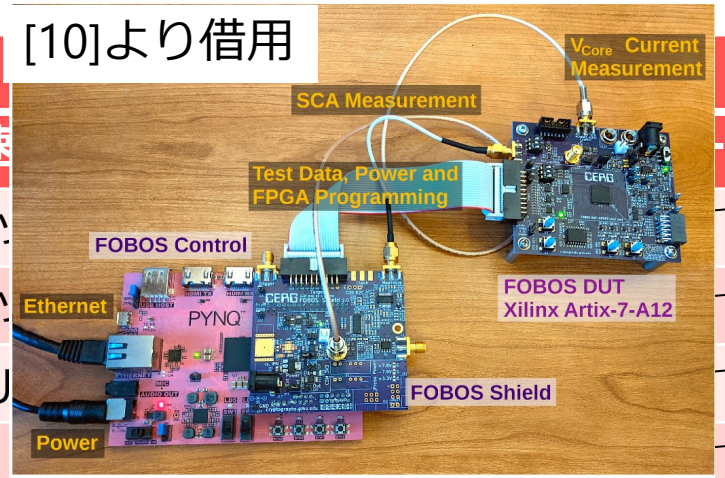
フレームワーク/文献	ソフトウェア			ハードウェア	
	予測モデル	実装	解析の高速化	ターゲット	トレース取得方法
RamDPA	S-Box出力	C++	マルチスレッド		
DareDevil	LUTファイル	C++	マルチスレッド		
T. Swamy, *et al.*	最終Round	C++	OpenCL/CUDA		
H. Gamaarachchi, *et al.*	最終Round		CUDA		
FOBOS3	別途計算	Python		FOBOS DUT	FOBOS Shiled
SASEBO/SAKURA					プ
ChipWhisperer					er-

- マルチスレッド、GPUの活用はあるもののデータの入力形式が独自
- 利用可能な予測モデルに制約

サイドチャネル攻撃研究用フレームワーク

フレームワーク/文献	ソフトウェア				
	予測モデル	実装	解析の高速		ース取得方法
RamDPA	S-Box出力	C++	マルチスレッ		
DareDevil	LUTファイル	C++	マルチスレッ		
T. Swamy, *et al.*	最終Round	C++	OpenCL/CU		
H. Gamaarachchi, *et al.*	最終Round		CUDA		
FOBOS3	別途計算	Python		FOBOS DUT	FOBOS Shiled
SASEBO/SA			マルチスレッド	SASEBO/SAKURA	VISA互換
ChipWhisp					

[10]より借用

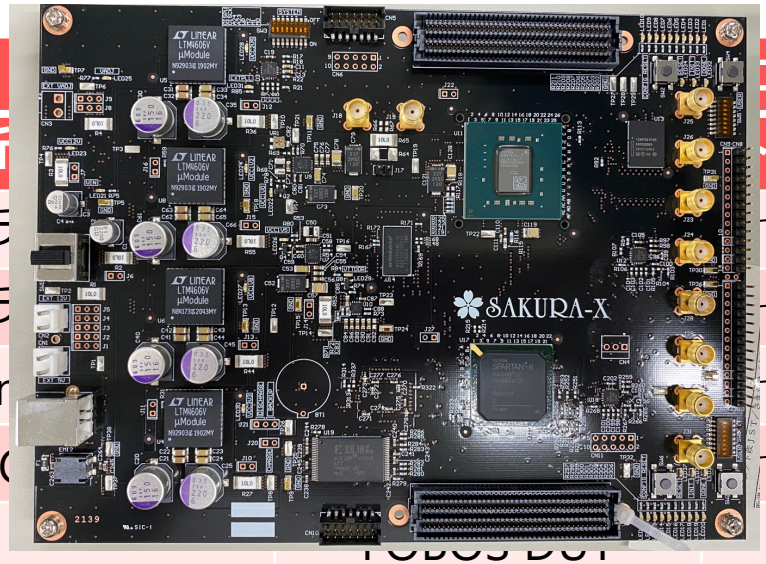


- SW/HWの両方がオープンソース
- PYNQ-Z2をコントローラとして利用
- ターゲットボードの基板設計データも公開 (市販はなし)
- 解析ツールがPython実装のため低速

[10] Ferrufino, Eduardo, et al. "FOBOS 3: An Open-Source Platform for Side-Channel Analysis and Benchmarking." Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security. 2023.

サイドチャネル攻撃研究用フレームワーク

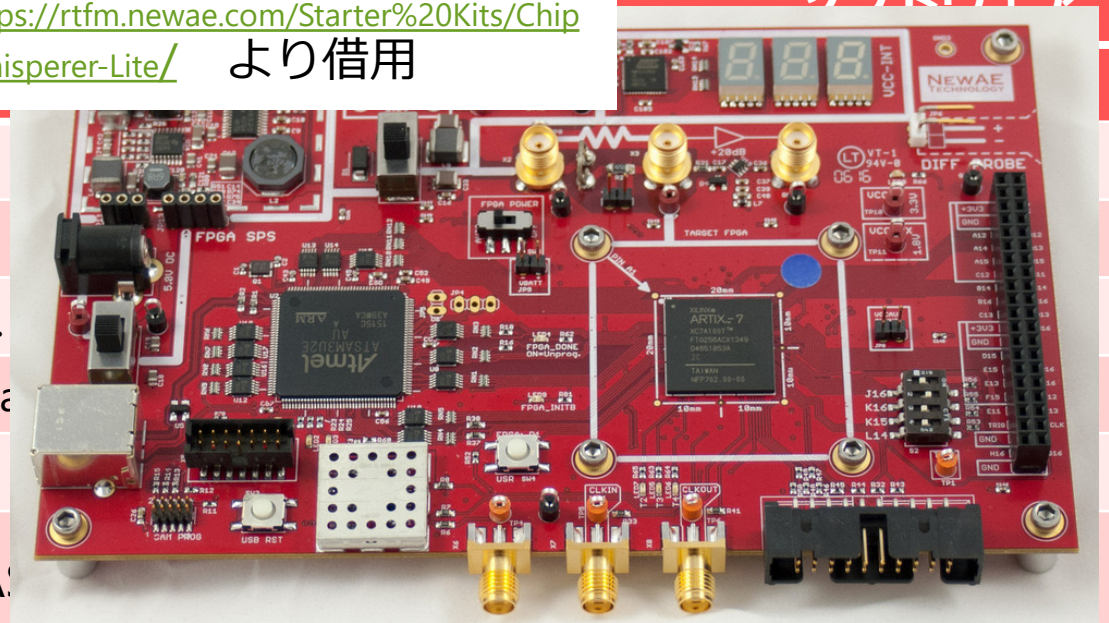
フレームワーク/文献	ソフトウェア			ハードウェア	
	予測モデル	実装	解析		トレース取得方法
RamDPA	S-Box出力	C++	マルチ		
DareDevil	LUTファイル	C++	マルチ		
T. Swamy, *et al.*	最終Round	C++	Oper		
H. Gamaarachchi, *et al.*	最終Round				
FOBOS3	別途計算	Python			FOBOS Shiled
SASEBO/SAKURA	最終Round	C#	マルチスレッド	SASEBO/SAKURA シリーズ	VISA互換 オシロスコープ
ChipWhisperer	利用可能	Python+ Jupyter	n/a	CWシリーズ	ChipWhisperer-Nano, etc



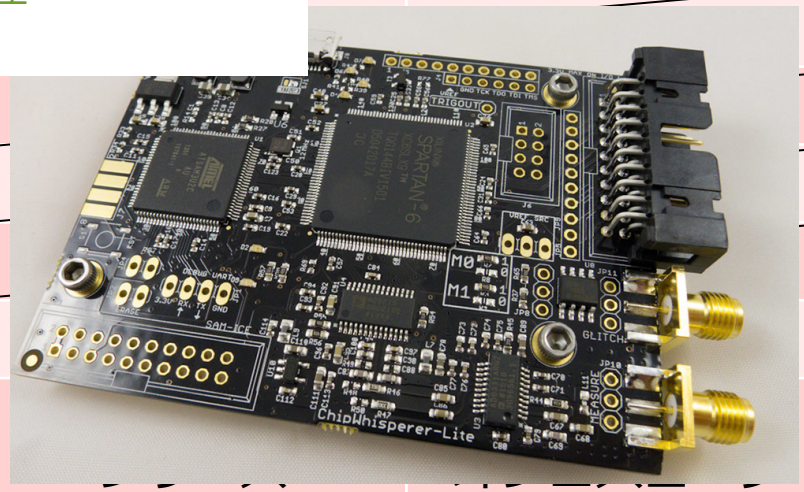
- SASEBOプロジェクトと後継のSAKURAプロジェクトからいくつかのボードが提供（DPAコンテストに採用された実績もあり）
- C#のツールも提供されているがWindowsでの利用が前提

サイドチャネル攻撃研究用フレームワーク

CW305ボード(Xilinx Artix-7)
https://rtfm.newae.com/Starter%20Kits/ChipWhisperer-Lite/ より借用



ChipWhisperer-Lite
10bit ADCなどを含む
https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA/
より借用



ChipWhisperer	複数利用可能	Python+ Jupyter	n/a	CWシリーズ	ChipWhisperer-Lite, etc

- 統一したプロジェクト保存形式
- 複数のリーク予測モデルで解析可能
- ターゲットデバイス、波形取得ボードなど多くの市販ボードをサポート
- Python実装のため複数プラットフォームで利用できるものの解析は低速

サイドチャネル攻撃研究用フレームワーク

フレームワーク/文献	ソフトウェア			ハードウェア	
	予測モデル	実装	高速化	ターゲット	トレース取得方法
RamDPA	S-Box出力	C++	マルチスレッド		
DareDevil	LUTファイル	C++	マルチスレッド		
T. Swamy, *et al.*	最終Round	C++	OpenCL/CUDA		
H. Gamaarachchi, *et al.*	最終Round		CUDA		
FOBOS3	別途計算	Python		FOBOS DUT	FOBOS Shiled
SASEBO/SAKURA	最終Round	C#	マルチスレッド	SASEBO/SAKURA シリーズ	VISA互換 オシロスコープ
ChipWhisperer	複数利用可能	Python+ Jupyter	n/a	CWシリーズ	ChipWhisperer-Nano, etc
提案フレームワーク	**ChipWhisperer に互換**	**Python C++**	**OpenMP OpenCL, CUDA**	**設計テンプレート**	**VISA互換 オシロスコープ**

サイドチャネル攻撃研究用フレームワーク

フレームワーク/文献	ソフトウェア			ハードウェア	
	予測モデル	実装	高速化	ターゲット	トレース取得方法
Ra					
T. S					
H. Gan					
					iled
SASI					換 ープ
Chip					perer- Nano, etc
提案フレームワーク	**ChipWhisperer に互換**	**Python C++**	**OpenMP OpenCL, CUDA**	**設計テンプレート**	**VISA互換 オシロスコープ**

- 提案フレームワークをChipWhispererのプラグインとして実装
- 相関係数解析をOpenMP, OpenCL, CUDAで高速化
  - Pybind11によるChipWhisperrerと同じインターフェース
    →既存のChipWhisperrerプロジェクトに対しても利用可能
  - 主要4ベンダー(Intel, AMD, NVIDIA, Apple)のGPUをサポート
- SAKURA-Xボードの向けのテンプレートを用意
  - VivadoのブロックデザインでAXIを持つ
  - ホストPCと鍵、平文などをやりとりする部分などが実装済み

新たなフレームワークの提案

CPA高速化ライブラリの実装

- 基本実装
  - C++で各種計算を実装、for文にOpenMPプラグマを挿入
  - 先に平均を求めるのではなくインクリメンタル形式で実装(部分トレースで計算可能)
- CUDA,OpenCL実装
  - 前述の内積計算とサンプル、予測リーケージの和,2乗和の計算をオフロード
    - 和、2乗和は再利用可能なのでGPUメモリ上で保持
  - 予測リーケージの計算と最終的な相関係数の計算はCPU(OpenMP)
- ChipWhispererの*AlgorithmsBase*を継承したPythonクラスを作成し、pybind11を用いて上記の実装をwrap
- 精度の問題
  - ChipWhispererの実装はnp.longdouble型
    →桁落ちが発生しうる箇所のみ拡張倍精度を採用
  - Apple SiliconのCPUは拡張倍精度がない
  - 一部のGPUはFP32までしかサポートしない
    - Intel Arc, Apple Silicon GPU

    →2変数を用いた高精度計算をエミュレートを実装[13]

インクリメンタル形式の式

$$\frac{N\sum_{i=1}^{N} h_{i,\hat{k}} w_{i,t} - \sum_{i=1}^{N} h_{i,\hat{k}} \sum_{i=1}^{N} w_{i,t}}{\sqrt{(\sum_{i=1}^{N} h_{i,\hat{k}})^2 - N\sum_{i=1}^{N} h_{i,\hat{k}}^2}\sqrt{(\sum_{i=1}^{N} w_{i,t})^2 - N\sum_{i=1}^{N} w_{i,t}^2}}$$

桁落ちが発生しやすい箇所

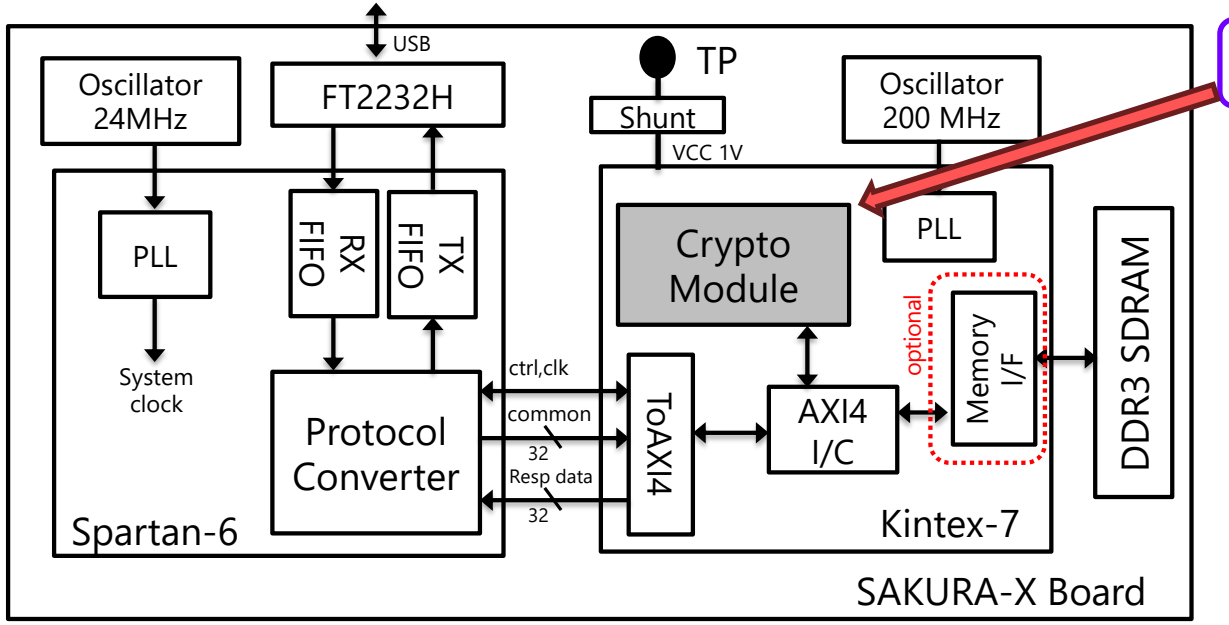
[13] Thall, Andrew. "Extended-precision floating-point numbers for GPU computation." ACM SIGGRAPH 2006 research posters. 2006. 52-es.

利用方法の例

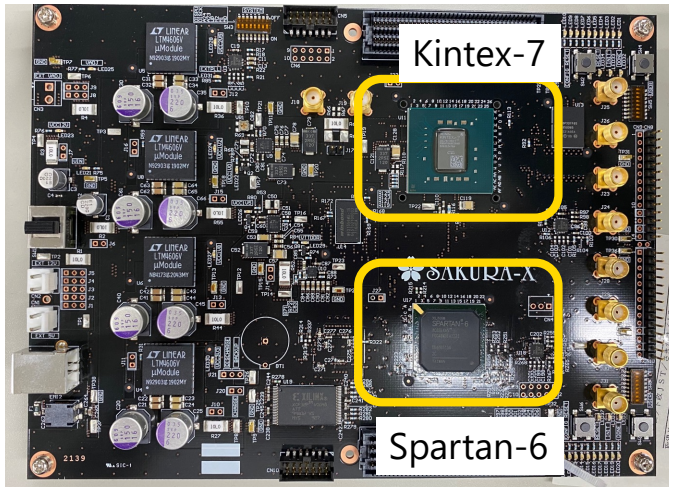
- 提案プラグインはpipでインストール可能
  - CMakeで高速化実装は自動的にビルド＆インストール
- CPAのインスタンス生成時に渡す引数で実装を切り替え

```
import chipwhisperer as cw
import chipwhisperer.analyzer as cwa
from cw_plugins.analyzer.attacks.cpa_algorithms.fast_progressive import FastCPAProgressiveCUDA
project = cw.open_project("project_name")    # プロジェクトデータを開く
leakage_model = cwa.leakage_models.sbox_output                    # リークモデル
attack = cwa.cpa(project, leakage_model, algorithm=FastCPAProgressiveCUDA) # GPU高速化実装を指定
result = attack.run()          # CPAの実行
```

SAKURA-X向け設計テンプレート



評価したいモジュールをここへ埋め込む



- Spartan-6: コントローラ部
  - ホストPCとUSBを介して通信
  - データ転送のためのコマンドを定義
  - プラグインにドライバを実装
- Kintex-7: 暗号モジュール部
  - AXIで接続可能
    →マルチクロックドメイン設計が可能
  - オプションでDRAMインターフェース(MIG)

設計サンプルとの比較

	提案テンプレート	SASEBOのサンプル	CW305のサンプル
ツール	Vivado	ISE	Vivado
ブロックデザイン	✓	×	×
インターフェース	AXI	独自	独自
DRAM	MIGのIP	×	×※

※: CW305にはDRAM未搭載、CW310には搭載されているものの設計サンプルなし

- 設計テンプレートは標準化されたインターフェースとブロックデザインをサポート
  - 高位合成によるコア、CPUのソフトコアなども容易に組み込み
  - IPコアが評価対象であればVerilog HDLの記述は不要

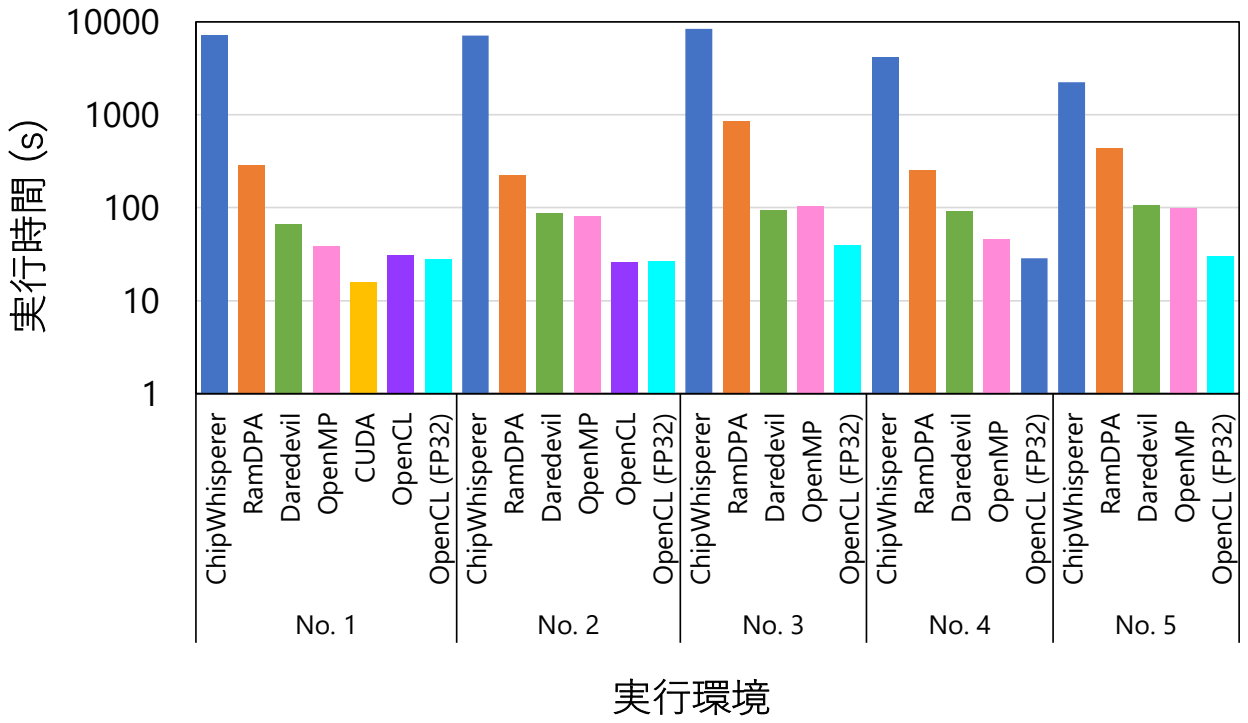
評価

CPA実行時間の評価

- オリジナルのChipWhisperer(Python), DareDevil, RamDPAとの比較
  - マルチスレッド処理は論理コア数で実行
- SAKURA-Xから取得した20000トレース, 4us区間(20000サンプル)
  - 使用オシロスコープ: Keysight Infiniviion MSO-X 4104A 5GS/s
- 以下の5つの環境で実行時間を各々測定

	CPU	RAM	GPU	GPU Toolkit/Runtime
No. 1	AMD Ryzen 7950X (16C32T)	128 GB	NVIDIA RTX-4070	CUDA 12.2
No. 2	Intel Core i9-14900KF (24C32T)	128 GB	AMD Radeon 7900 XTX	ROCm 6.0.1
No. 3	AMD Ryzen 5700G (8C16T)	64 GB	Intel ARC A770	Intel graphics compute runtime 23.22.26516.34
No. 4	Apple M1 Ultra (20C)	128 GB	64 cores GPU	Xcode 14.2
No. 5	Apple M2 Max (12C)	96 GB	38 cores GPU	Xcode 15.2

CPAの実行時間比較

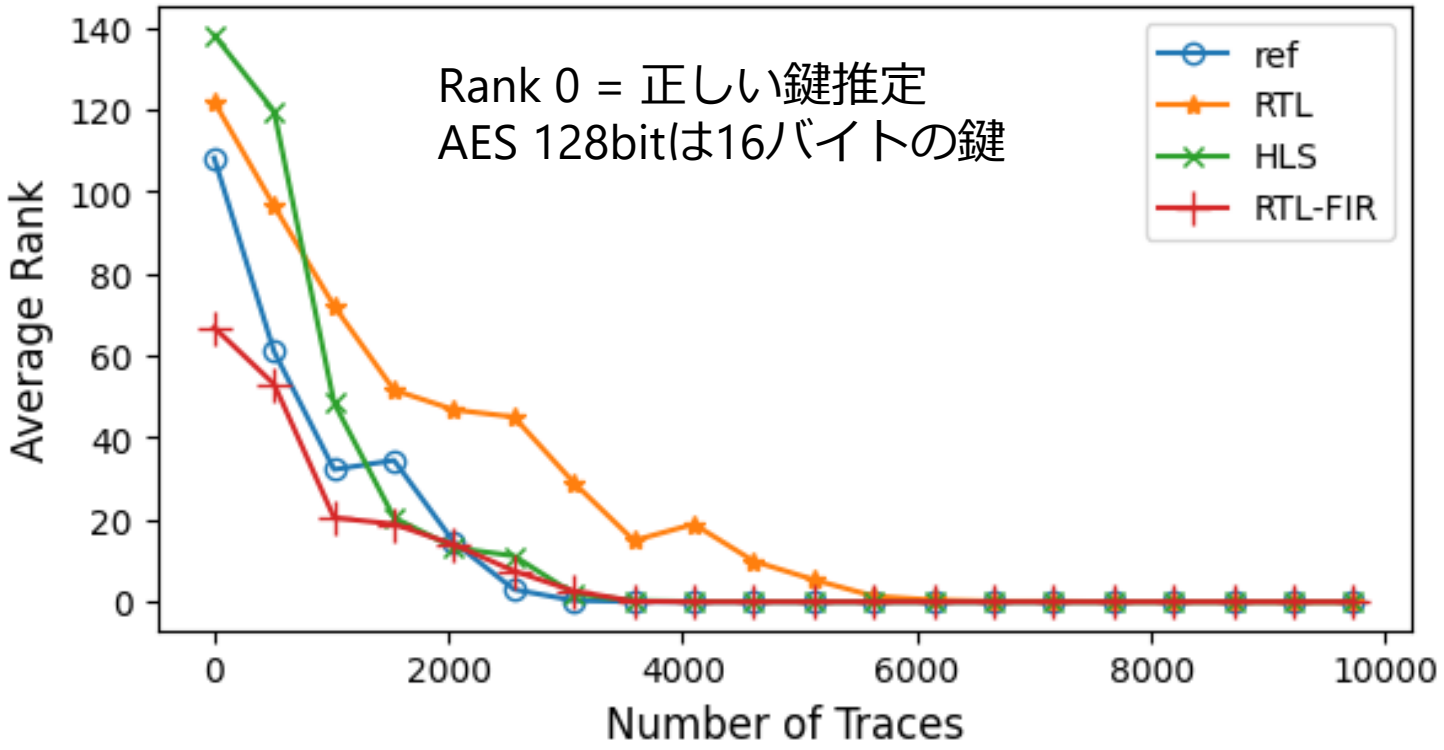


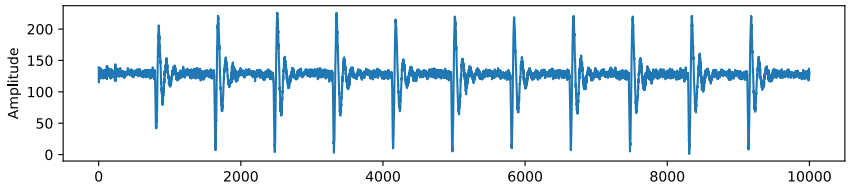
- 数時間を要するChipWhisperの実装から提案OpenMP実装は40s-100s程度に短縮
  - 既存のマルチスレッド実装のRamDPA,DareDevilよりも数倍高速なケースも確認
- GPU活用によりさらに2~3倍の高速化を達成
  - ChipWhisperer実装(7043s)と比べCUDA実装(15.7s)で450倍の高速化

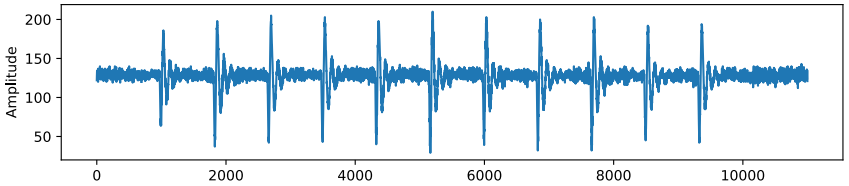
設計テンプレートの評価

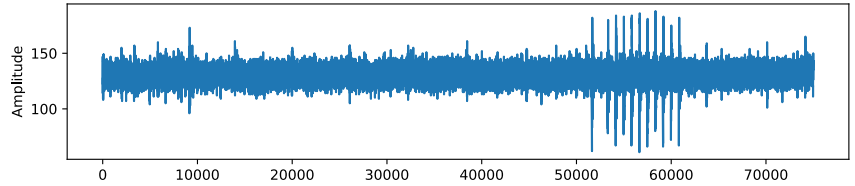
- AES暗号モジュールをいくつか設計
  1. SASEBOのサンプル (比較用, RTL実装)
  2. 設計テンプレート+RTL実装AESコア(AXI slave)
  3. 設計テンプレート+HLS実装AESコア
     (AXI Master-AXI BRAM Controller-Block Memory Generator)
  - AES演算コアについてはいずれもガロア拡大体によるS-Box実装
- 動作周波数
  - AESコア: 6MHz (SASEBOのサンプルと揃えるため)
  - 設計テンプレートのコントローラ、AXI変換部 100MHz
- ツール: ISE 14.7, Vivado 2023.2, Vitis 2023.2
- 設計テンプレートのリソース消費 (Kintex-7 xc7k160t)
  - LUT: 141 (0.139%), FF: 594 (0.292%)

設計テンプレートの有無による影響





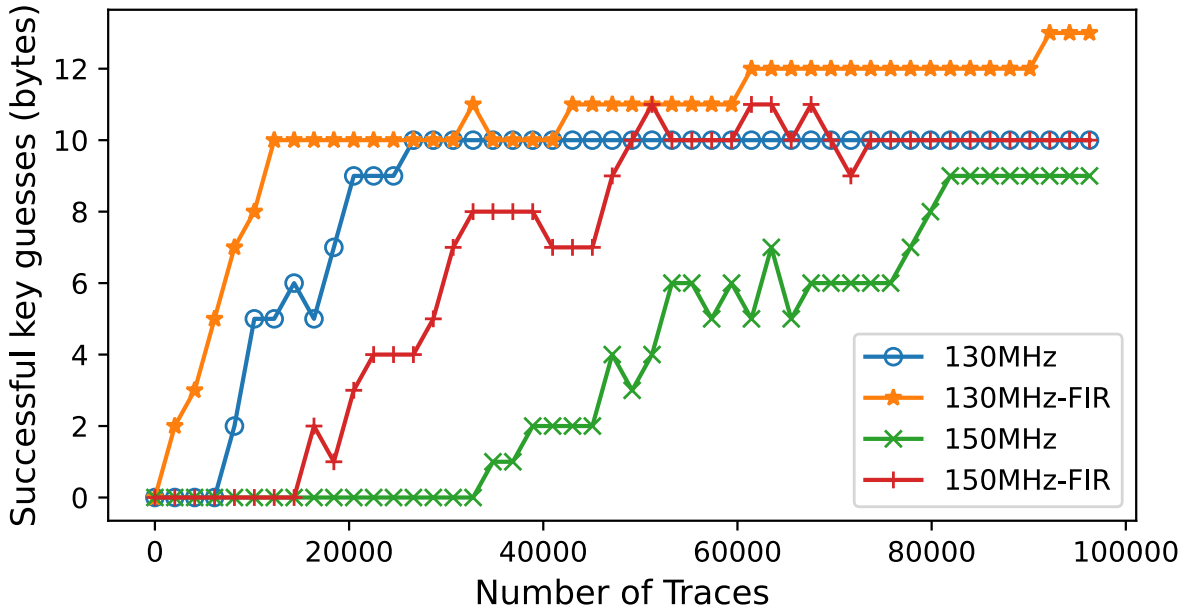




- SASEBOのサンプル(テンプレートなし)とテンプレートを用いたRTL実装とHLS実装を同じ6MHzで動作させて波形を取得
  - 各ラウンドを1サイクルで計算する
  - HLS実装ではトリガーとなる信号をBRAMへのアクセスに関連した信号
- 概ね同じトレース数で解析に成功する→テンプレートの影響は小さい

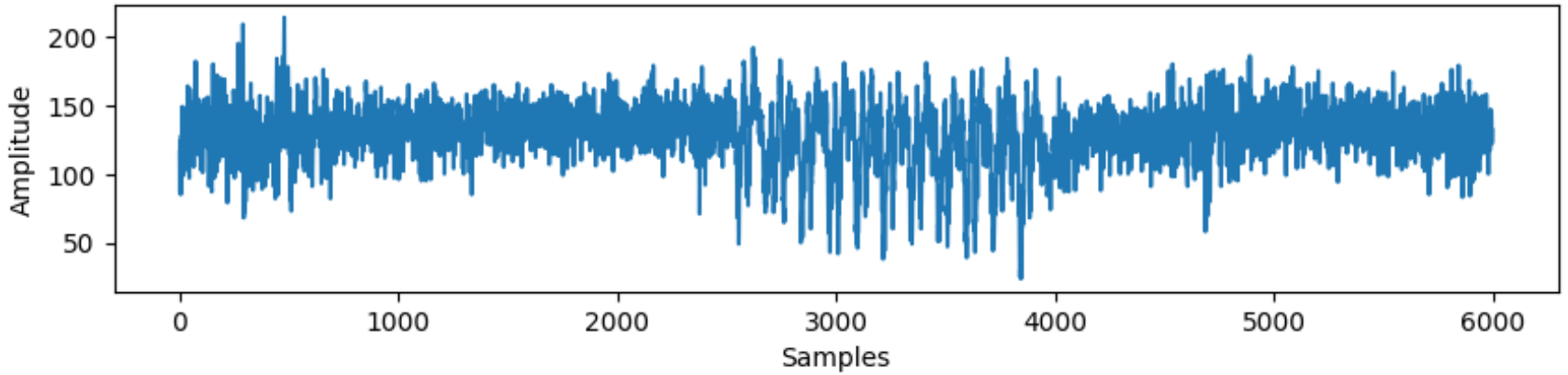
HLSツール最適化の影響

- タイミング制約に基づき自動的に演算サイクル数を最適化
  - 動作周波数が向上する一方でレジスタへの中間データ書き込み回数が増加し新たなリーク源となりうる可能性
- 前述のHLS設計
  - ~50MHz: 1 cycle/round
  - 60~110MHz: 2 cycles/round
  - 120MHz~140MHz: 3 cycles/round
  - 150MHz: 5 cycles/round

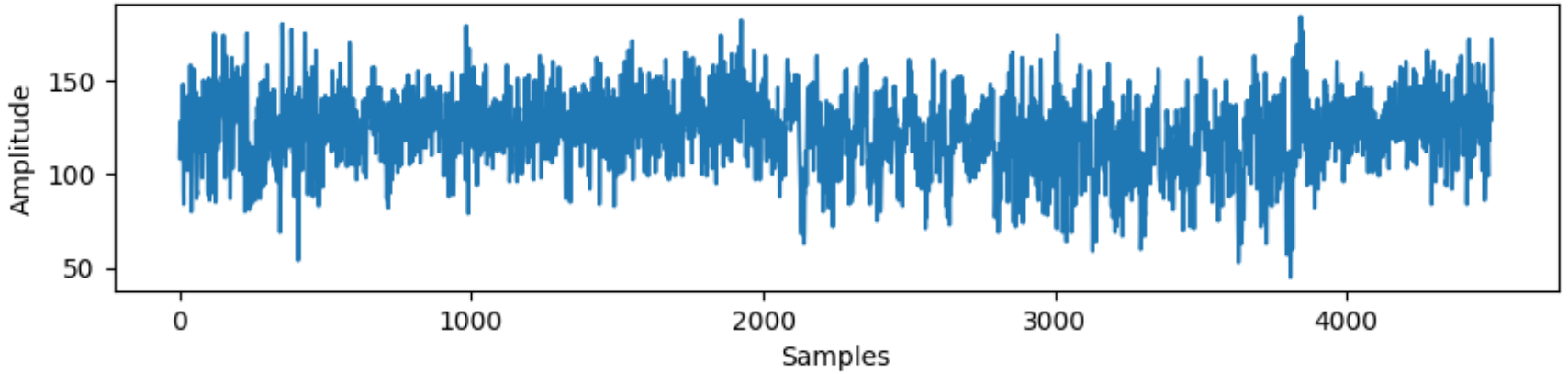


- 120MHz, 150MHzのトレースを1000トレース取得
  - いずれも一部の部分鍵(9~13バイト)しか推定に成功せず
  - ただし、リセットされない中間データ用レジスタの存在は確認

HLS実装の電力波形



120MHz (10ラウンド 30cycles)



150MHz (10ラウンド 50cycles)

まとめ

- IoT端末の普及でハードウェアセキュリティの重要性が高まる
- サイドチャネル攻撃は暗号処理モジュールなどの内部状態(鍵など)を不正に入手する攻撃として大きな脅威
- 耐サイドチャネル攻撃の暗号処理モジュール設計にかかる時間的コストが問題
  - 消費電力波形の取得環境
  - 膨大な取得データの解析にかかる時間
- 提案: 広く利用されるフレームワークのプラグインを開発
  - 標準的に利用されるFPGAボード向けの設計テンプレートとそのドライバ
  - 時間のかかる相関係数解析をマルチスレッド、GPUによって高速化
- ケーススタディ
  - 高位合成によるAES計算コアに対するサイドチャネル攻撃を評価
  - 高位合成ツールの最適化が新たな脆弱性を生み出すことは現時点では未確認
- 今後の展望
  - プラグインを高次CPAへの拡張
  - ソフトコアGPU(e.g., Vortex)などを用いた機械学習へのサイドチャネル攻撃評価