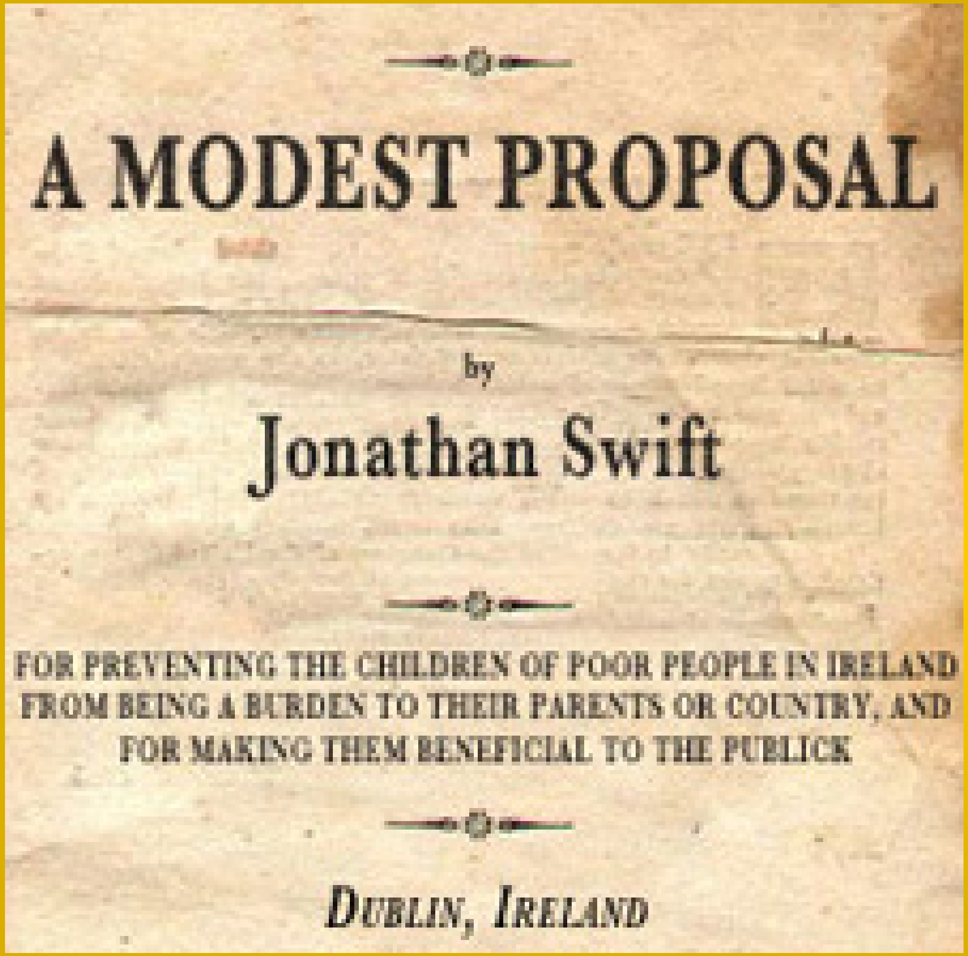
SWIFFT: A Modest Proposal For FFT Hashing

Vadim Lyubashevsky	UCSD
Daniele Micciancio	UCSD
Chris Peikert	SRI International
Alon Rosen	Herzliya IDC



A MODEST PROPOSAL

by

Jonathan Swift

FOR PREVENTING THE CHILDREN OF POOR PEOPLE IN IRELAND
FROM BEING A BURDEN TO THEIR PARENTS OR COUNTRY, AND
FOR MAKING THEM BENEFICIAL TO THE PUBLICK

Dublin, Ireland

SWIFFT

A collection of compression functions

- Efficient
- Highly parallelizable } FFT
- Supporting proof of security

Not an "all-purpose" function

- In particular, it is linear: f(x+y) = f(x) + f(y)
- However, has many desirable properties
    a) Cryptographic
    b) Statistical

Our Starting Point

At a (very) high-level:

- ☐ **Key**: m random deg < n polynomials in α
- ☐ **Input**: m polynomials w/ binary 0-1 coefficients
- ☐ **Function**: compute sum of products

All arithmetic modulo p and $\left(\alpha^n + 1\right)$

$$R = \mathbb{Z}_p[\alpha] / \left(\alpha^n + 1\right)$$

- ☐ **Key** : $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m \in R$
- ☐ **Input**: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \{0,1\}^n \subset R$
- ☐ **Function**: $f_{\mathbf{A}}(\mathbf{X}) = \sum_{i=1}^{m} \mathbf{a}_i \cdot \mathbf{x}_i \in R$

Supporting Proof of Security

For random key $\mathbf{A}$, the function is collision resistant, assuming worst-case hardness in cyclic/ideal lattices [PR06, LM06].

- ☐ Continues a long line of works [Ajtai96,…,Mic02]
- ☐ proof is asymptotic
- ☐ meaningful only for large parameters

In this work:

- ☐ Concrete parameters (m=16, n=64, p=257)
- ☐ Function maps 1024 bits to 528 bits
- ☐ Very efficient implementation.

- ☐ Security proof suggests that design is sound
- ☐ Heuristic analysis suggests that parameters are sound

Towards Efficient Implementation

Central Observations:

1. Polynomial multiplication ⇔ FFT

$$\mathbf{a}_i \cdot \mathbf{x}_i = \mathrm{FFT}^{-1}(\mathrm{FFT}(\mathbf{a}_i) \odot \mathrm{FFT}(\mathbf{x}_i))$$

2. Can pre-compute $\mathrm{FFT}(\mathbf{a}_1), \mathrm{FFT}(\mathbf{a}_2), \ldots, \mathrm{FFT}(\mathbf{a}_m)$

3. No need to compute FFT^{-1}

4. Specifics of $\mathbb{Z}_p[\alpha]/\left(\alpha^n + 1\right)$ allow FFT optimization
    a) Can perform *modular* FFT (NTT)
    b) FFT of dimension n is sufficient

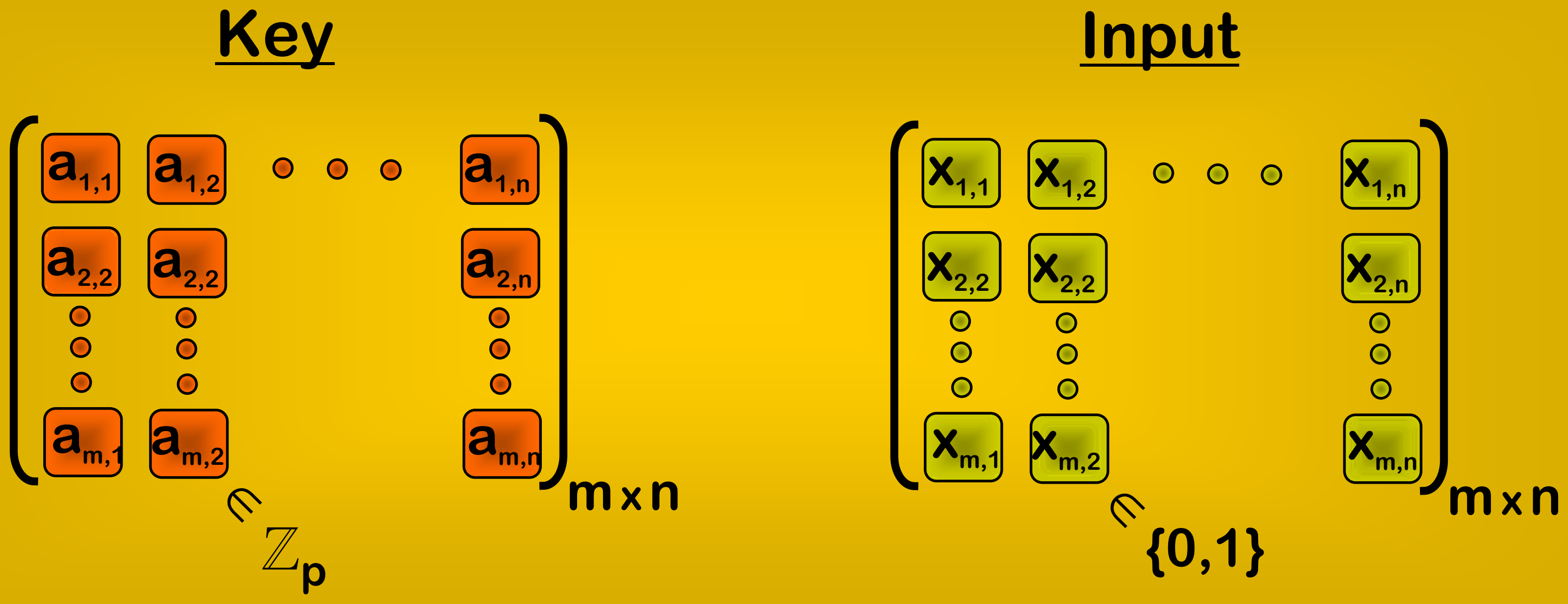
Resulting function is completely equivalent (security-wise).

SWIFFT

Parameters: n, m, p

Key: $m \times n$ matrix $\left(a_{i,j}\right) \in \mathbb{Z}_p^{m\times n}$

Input: $m \times n$ binary matrix $\left(x_{i,j}\right) \in \{0,1\}^{m\times n}$



Key

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,2} & a_{2,2} & & a_{2,n} \\ \vdots & \vdots & & \vdots \\ a_{m,1} & a_{m,2} & & a_{m,n} \end{pmatrix}_{m \times n} \quad \in \mathbb{Z}_p$$

Input

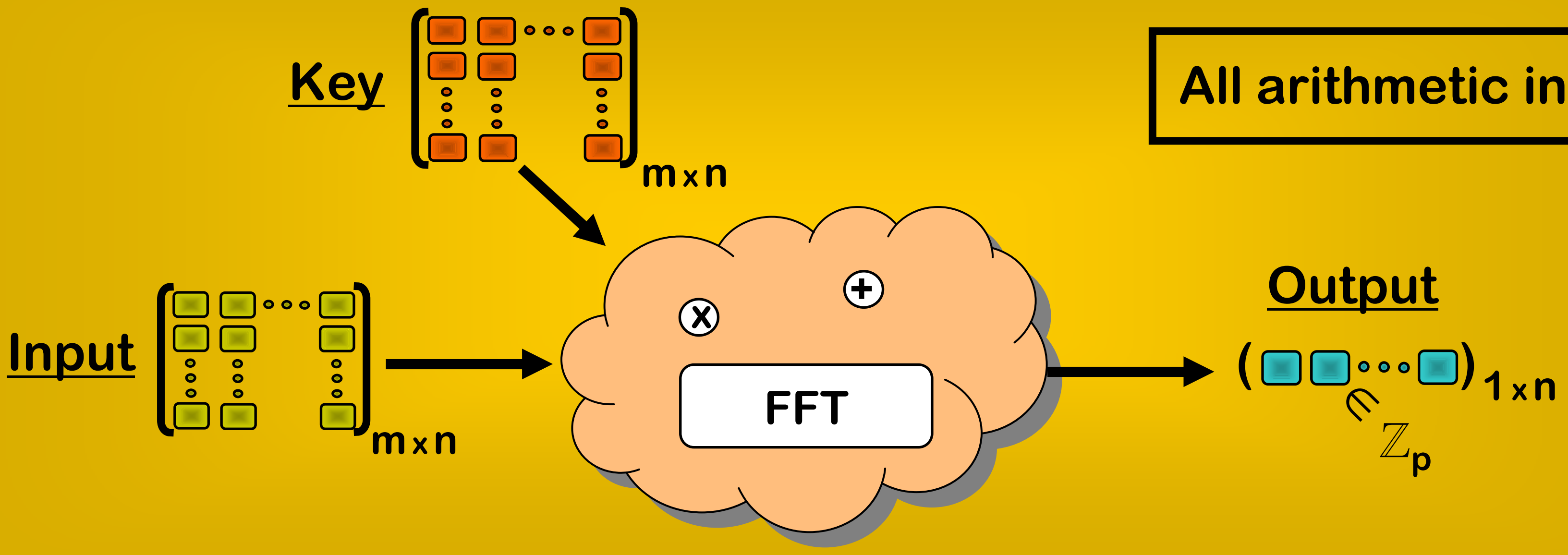
$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,2} & x_{2,2} & & x_{2,n} \\ \vdots & \vdots & & \vdots \\ x_{m,1} & x_{m,2} & & x_{m,n} \end{pmatrix}_{m \times n} \quad \in \{0,1\}$$

SWIFFT

Parameters: n, m, p

Key: $m \times n$ matrix $\left(a_{i,j}\right) \in \mathbb{Z}_p^{m \times n}$

Input: $m \times n$ binary matrix $\left(x_{i,j}\right) \in \{0,1\}^{m \times n}$



Key

$m \times n$

Input

$m \times n$

FFT

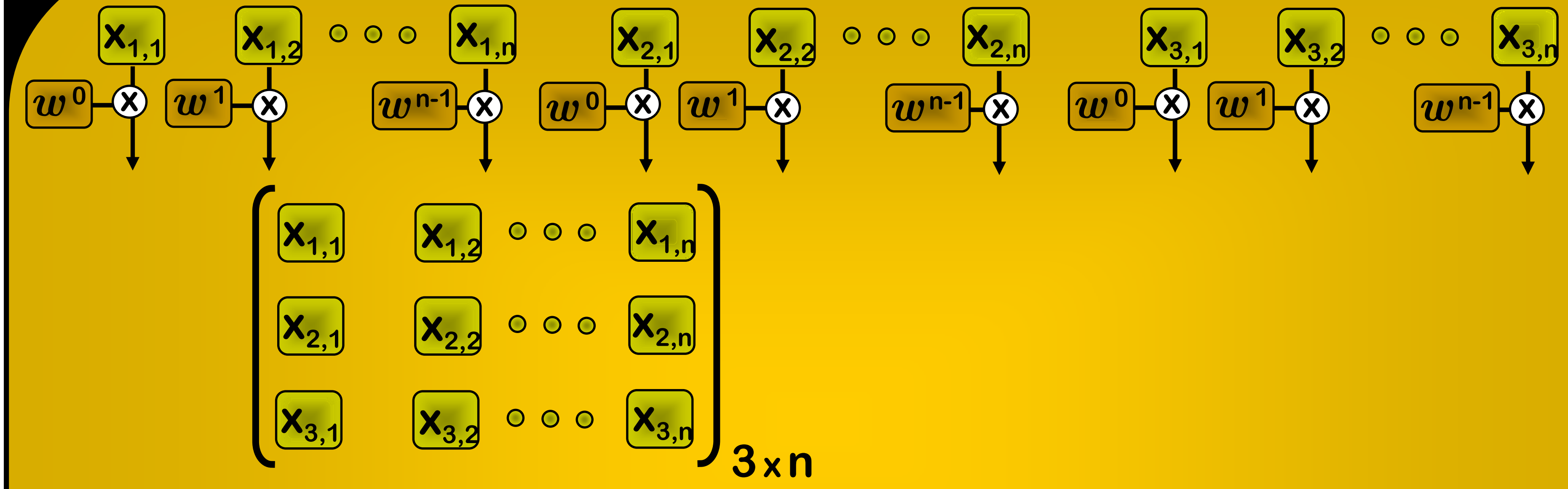
Output

$1 \times n$

$\in \mathbb{Z}_p$

All arithmetic in $\mathbb{Z}_p$

SWIFFT (m=3)

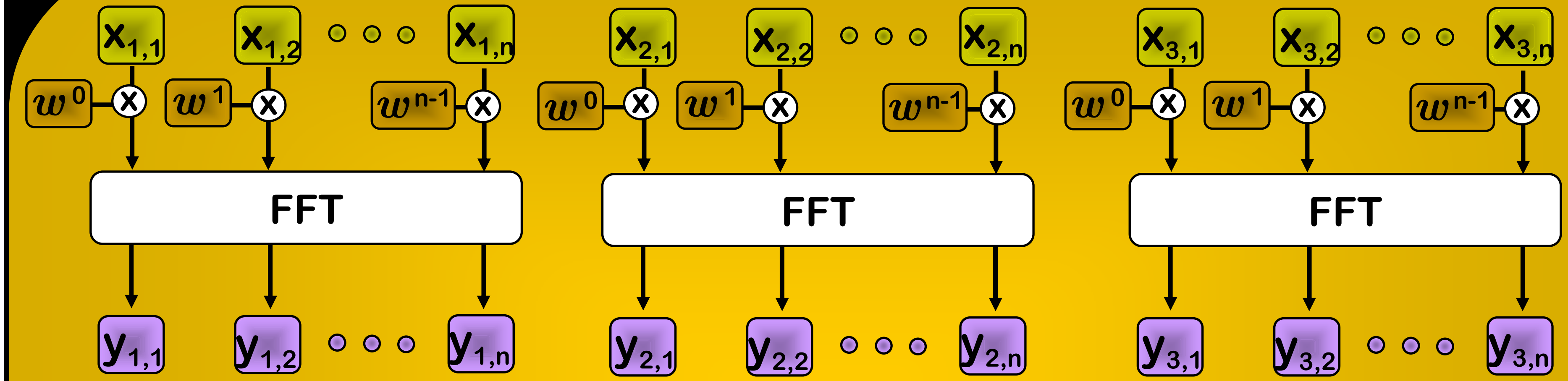


Step 1: For each row $i = 1, \ldots, m$ compute:

$$\left(y_{i,1}, \ldots, y_{i,n} \right) = \mathrm{FFT}(\omega^0 \cdot x_{i,1}, \ldots, \omega^{n-1} \cdot x_{i,n})$$

where $\omega \in \mathbb{Z}_p$ is a 2nth root of unity in $\mathbb{Z}_p$

SWIFFT (m=3)

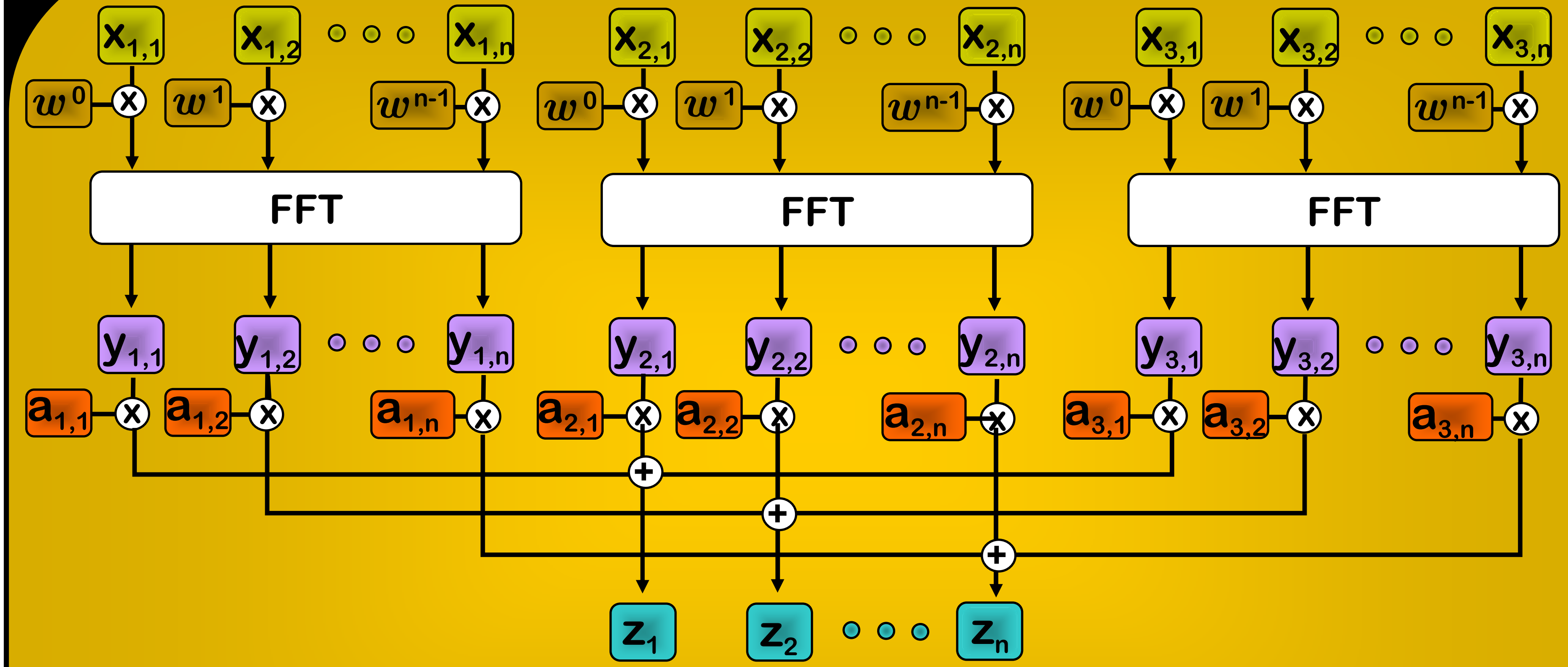


Step 1: For each row $i = 1, \ldots, m$ compute:

$$\left(y_{i,1}, \ldots, y_{i,n} \right) = \mathrm{FFT}(\omega^0 \cdot x_{i,1}, \ldots, \omega^{n-1} \cdot x_{i,n})$$

where $\omega \in \mathbb{Z}_p$ is a 2n^th root of unity in $\mathbb{Z}_p$

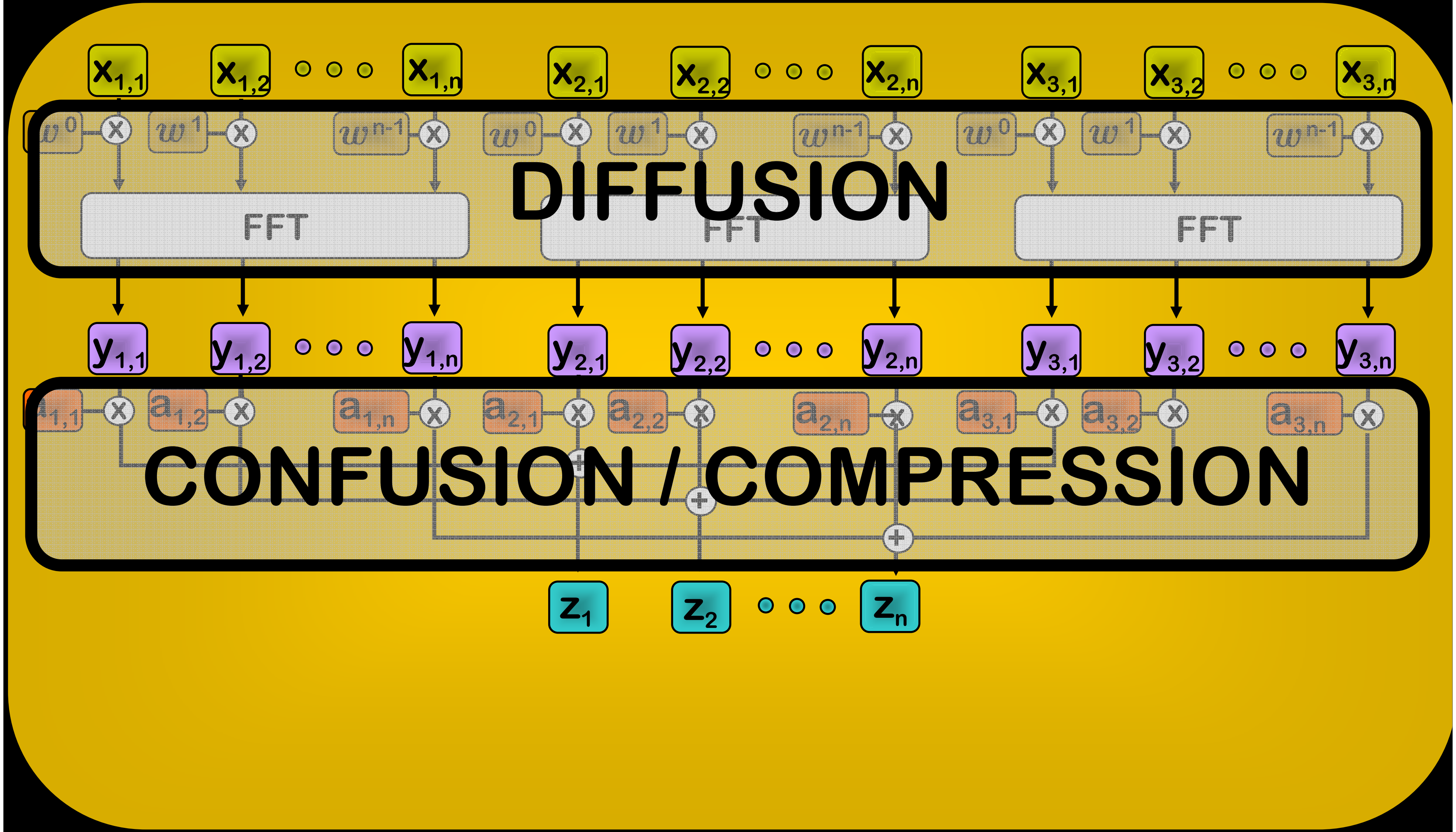
SWIFFT (m=3)



Step 2: For each column $j = 1, \ldots, n$ compute:

$$z_j = a_{1,j} \cdot y_{1,j} + \cdots + a_{m,j} \cdot y_{m,j}$$

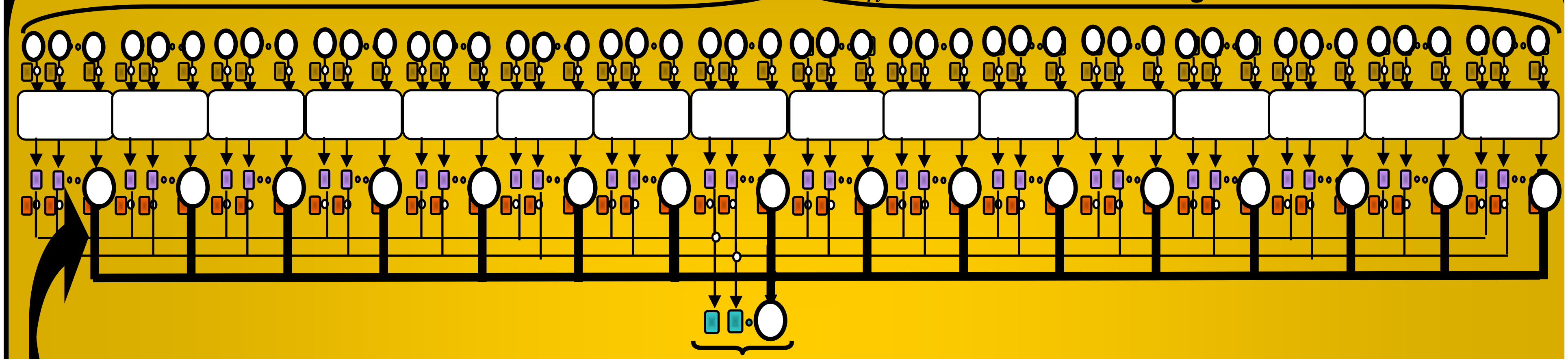
SWIFFT (m=3)


x1,1
x1,2
x1,n
x2,1
x2,2
x2,n
x3,1
x3,2
x3,n
w0
w1
wn-1
DIFFUSION
FFT
FFT
FFT
y1,1
y1,2
y1,n
y2,1
y2,2
y2,n
y3,1
y3,2
y3,n
a1,1
a1,2
a1,n
a2,1
a2,2
a2,n
a3,1
a3,2
a3,n
CONFUSION / COMPRESSION
z1
z2
zn

SWIFFT (m = 16, n = 64, p=257)

nm = 1024 bits

Hard to solve $z_1,\ldots,z_n$ simultaneously



n $\log_2$ p ~ 528 bits

Easy to find solution $\left(y_{1,j},\ldots,y_{m,j}\right)$ to each $z_j=\sum_{i=1}^{m} a_{i,j}\cdot y_{i,j}$ individually

- The reason: $\left(y_{i,1},\ldots,y_{i,n}\right)$ are highly constrained
  1. Dependency through FFT
  2. Need to find binary $\left(x_{i,1},\ldots,x_{i,n}\right)$
- This way of “breaking linearity” is different from previous proposals for FFT hashing [S91,S92,SV93,V92].

Choice of Parameters (m=16, n=64, p=257)

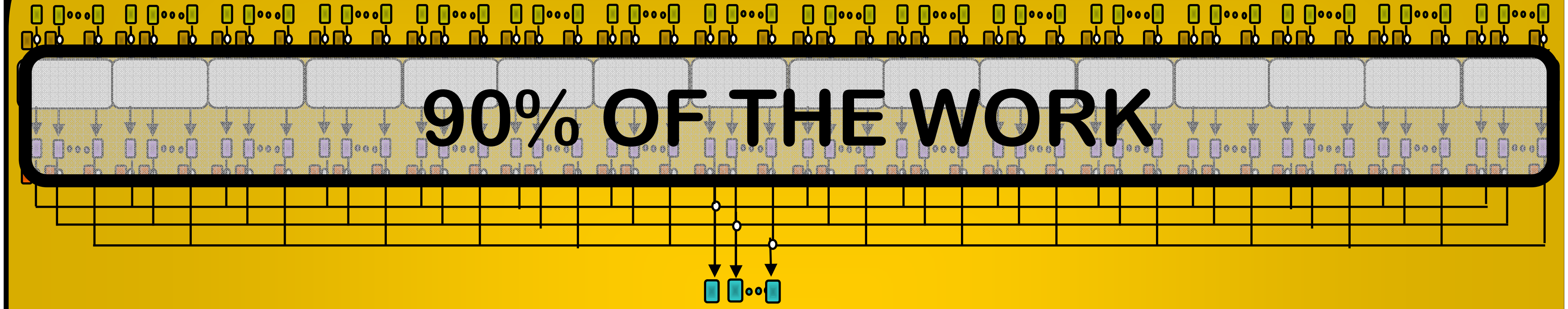
Security considerations:

- ☐ Subset-sum instance from 1024 to 528 bits.
- ☐ n = 2^k ⇔ modulus polynomial $\left(\alpha^n + 1\right)$ is irreducible (over $\mathbb{Q}$)
  1. crucial for security proof
  2. otherwise can find collisions (LASH, [Mic02] OWF).
  3. Enables to avoid straightforward weaknesses

Performance considerations:

- ☐ p=257 is a prime of the form p = 4n+1
- ☐ enables efficient 64-dimensional modular FFT.
  1. $\mathbb{Z}_{257}$ is a field
  2. $\omega \in \mathbb{Z}_{257}$ is a 128th root of unity
  3. Odd powers of w are the roots of $\left(\alpha^n + 1\right)$

Fast Implementation



To improve performance:

1. Use lookup tables inside FFT.
2. Parallelize atomic operations (+,x).
3. Avoid modular reductions (whenever possible).
4. Multiply by w powers using left shifts.

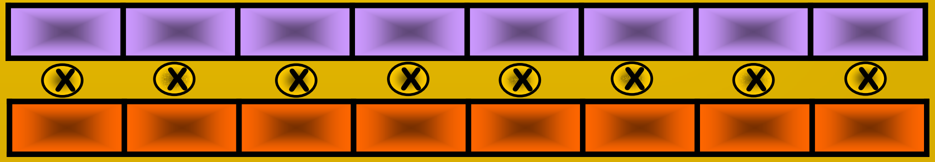
Speeding up the FFT

Input to FFT is a binary vector:

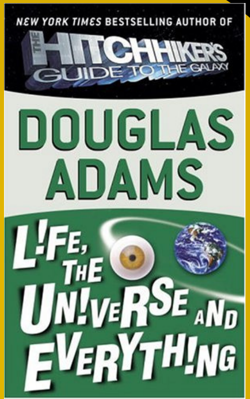
- ☐ Few possible intermediate values.
- ☐ Can pre-compute and store in lookup table.

FFT is highly parallelizable:

- ☐ Reduce FFT_{64} to 8 parallel FFT_8
- ☐ Use SIMD (single-instruction multiple-data) instructions to perform operations in parallel.
- ☐ Point-wise vector addition/multiplication on 8 dim. registers (w/ 16 bit signed integer entries).

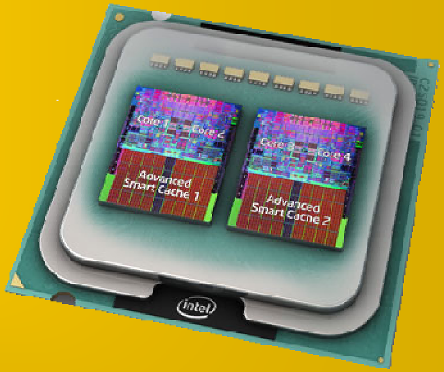


Further Optimizations



Use of $\mathbb{Z}_{257}$:

- $\omega = 42$ is a 128th root of unity mod 257.
- FFT$_8$ uses $\omega^{16} = 2^2$ (mod 257).
- Multiplications by powers of ω^{16} using left shifts.
- Can avoid most modular reductions w/out overflow.
- Use SIMD for parallel modular reduction.



Multi-core processors:

- FFTs are completely independent.
- We did not exploit multi-core capabilities yet

Performance

Implemented and tested:

- ☐ On 3.2 GHz Intel Pentium 4.
- ☐ Written in C (using INTEL intrinsics for SSE2).
- ☐ Compiled using gcc 4.1.2 on Linux kernel 2.6.19.

Compared to SHA256:

- ☐ Same system
- ☐ `openssl` version 0.9.8 speed benchmark

Results:

- ☐ **SWIFFT** - Throughput ~40 MB/s
- ☐ **SHA256** - Throughput ~47MB/s

Statistical/Cryptographic Properties

Statistical properties (no computational assumptions):

1. Universal hashing
2. Regularity
3. Randomness extraction

Cryptographic properties:

1. One-wayness
2. Second-preimage resistance
3. Target collision resistance
4. Collision resistance

We aim for 2^{100} security. We do NOT claim:

❑ 2^{528} security against inversion attacks

❑ $2^{528/2}$ security against collision attacks

Concrete Security Analysis

A convenient way to view SWIFFT is as a subset-sum instance:

$$\mathbf{a}\cdot\mathbf{x}\in R \quad \leftrightarrow \quad \begin{bmatrix} a_0 & -a_{n-1} & \cdots & -a_1 \\ a_1 & a_0 & & -a_2 \\ \vdots & & \ddots & \\ a_{n-1} & & \cdots & a_0 \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \bmod p$$

Our function can be viewed as multiplying a vector $\mathbf{x}\in\{0,1\}^{mn}$ with a matrix $\mathbf{A}=\left[\mathbf{A}_1 \mid \cdots \mid \mathbf{A}_m\right]_{n\times mn}$ where $\mathbf{A}_i$ is the skew-circulant matrix that corresponds to $\mathbf{a}_i$

- ❑ Best known attack:
  - Wagner’s generalized birthday attack.
  - Has complexity 2^{106}
- ❑ Lattice reduction algorithms do not do as well.

Conclusions

SWIFFT: FFT-based hashing

- ☐ Provably secure design (under worst-case assumption)
- ☐ Concrete instantiation w/ heuristic security analysis
- ☐ Highly efficient implementation

Future directions

- ☐ Further cryptanalysis (possibly algebraic)
- ☐ Faster implementation
- ☐ Shorter output/smaller description
- ☐ Exploiting linearity for applications