Faster Secure Two-Party Computation Using Garbled Circuits

Yan Huang
David Evans

Jonathan Katz
Lior Malka

www.MightBeEvil.com

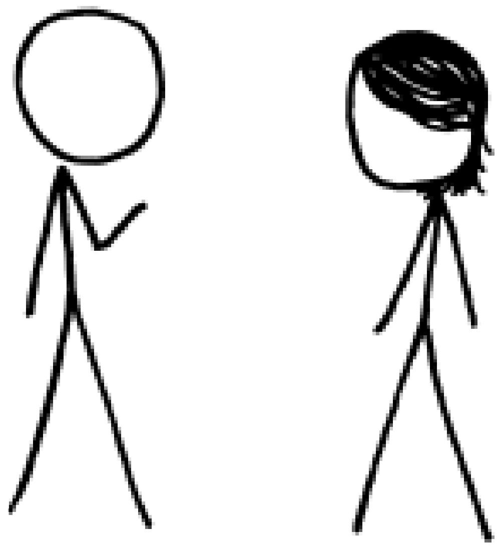
Secure Two-Party Computation

Bob's Genome: ACTG…
Markers (~1000): [0,1, …, 0]

Alice's Genome: ACTG…
Markers (~1000): [0, 0, …, 1]

Bob

Alice



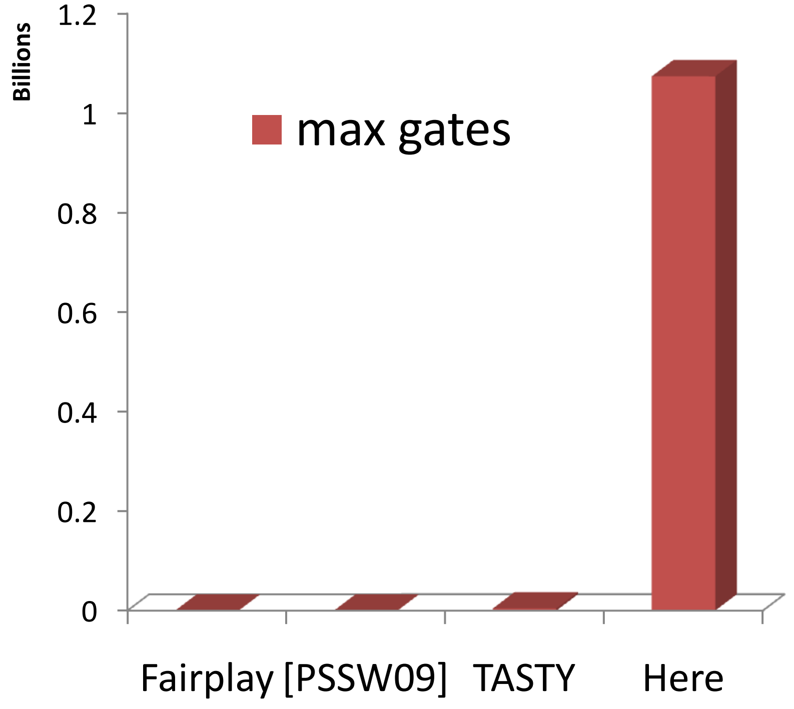
$$x = f(g_A, g_B)$$

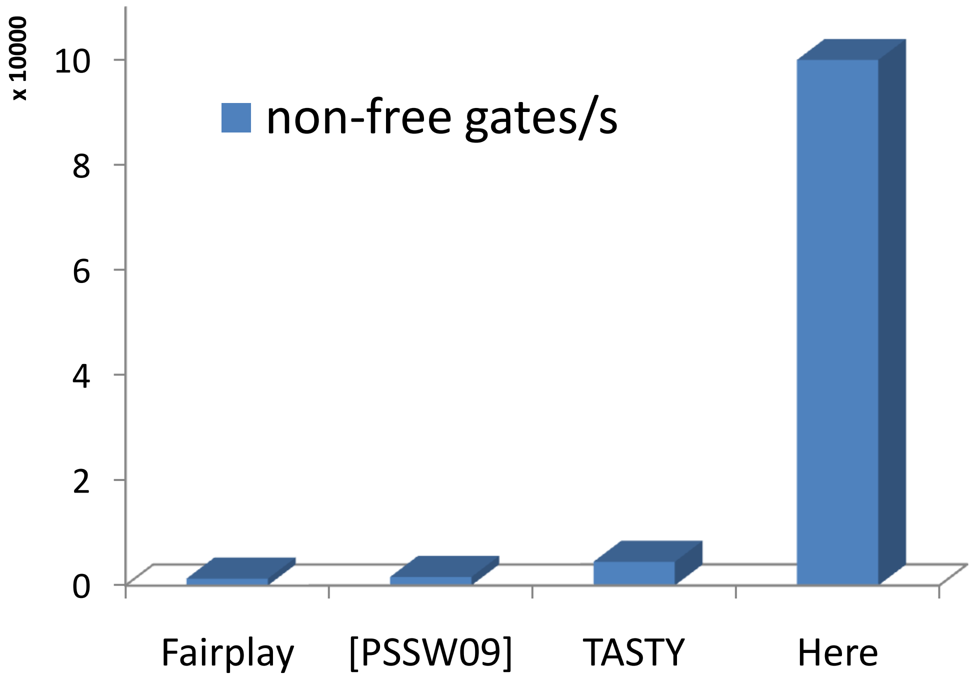
Can Alice and Bob compute a function of their private data, without exposing anything about their data besides the result?

Overview

- Describe a system for secure 2-party computation using garbled circuits that is much more ***scalable*** and significantly ***faster*** than best prior work
- Applications:
  - **Face recognition:** Hamming distance
  - **Genomics:** Edit distance, Smith-Waterman
  - **Private encryption:** Oblivious AES evaluation

Our Results





Scalability

Performance

Secure Function Evaluation

Alice (circuit generator)

Bob (circuit evaluator)

Holds $a \in \{0,1\}^s$

Agree on
$f(a, b) \rightarrow x$

Holds $b \in \{0,1\}^t$

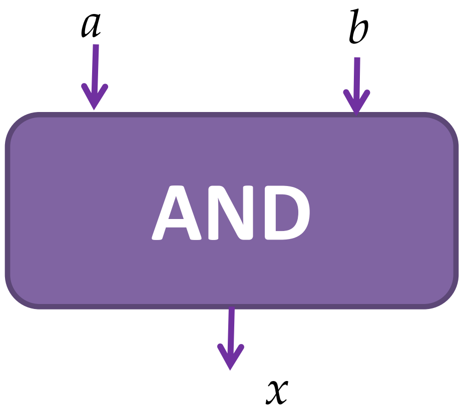
Garbled Circuit Protocol

Outputs $x = f(a, b)$
without revealing a
to Bob or b to Alice.

Andrew Yao, 1986

Yao's Garbled Circuits

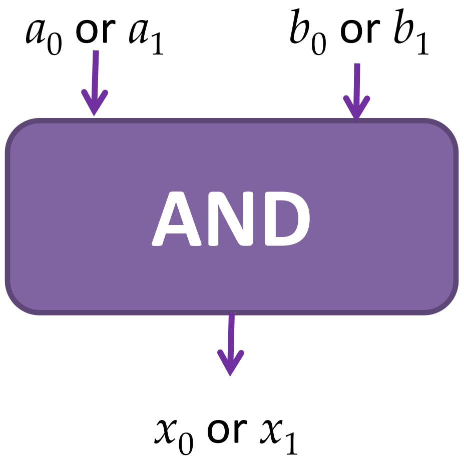
Inputs		Output
a	b	x
0	0	0
0	1	0
1	0	0
1	1	1



Computing with Meaningless Values?

Inputs		Output
a	b	x
a_0	b_0	x_0
a_0	b_1	x_0
a_1	b_0	x_0
a_1	b_1	x_1

a_i, b_i, x_i are **random** values, chosen by the **circuit generator** but **meaningless** to the **circuit evaluator**.



Computing with Garbled Tables

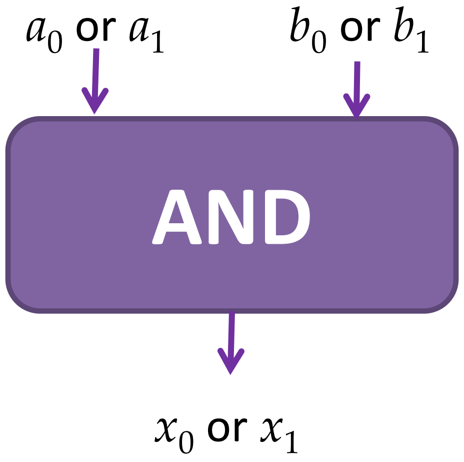
Inputs		Output
a	b	x
a_0	b_0	$Enc_{a0,b0}(x_0)$
a_0	b_1	$Enc_{a0,b1}(x_0)$
a_1	b_0	$Enc_{a1,b0}(x_0)$
a_1	b_1	$Enc_{a1,b1}(x_1)$

Bob can only decrypt **one** of these!

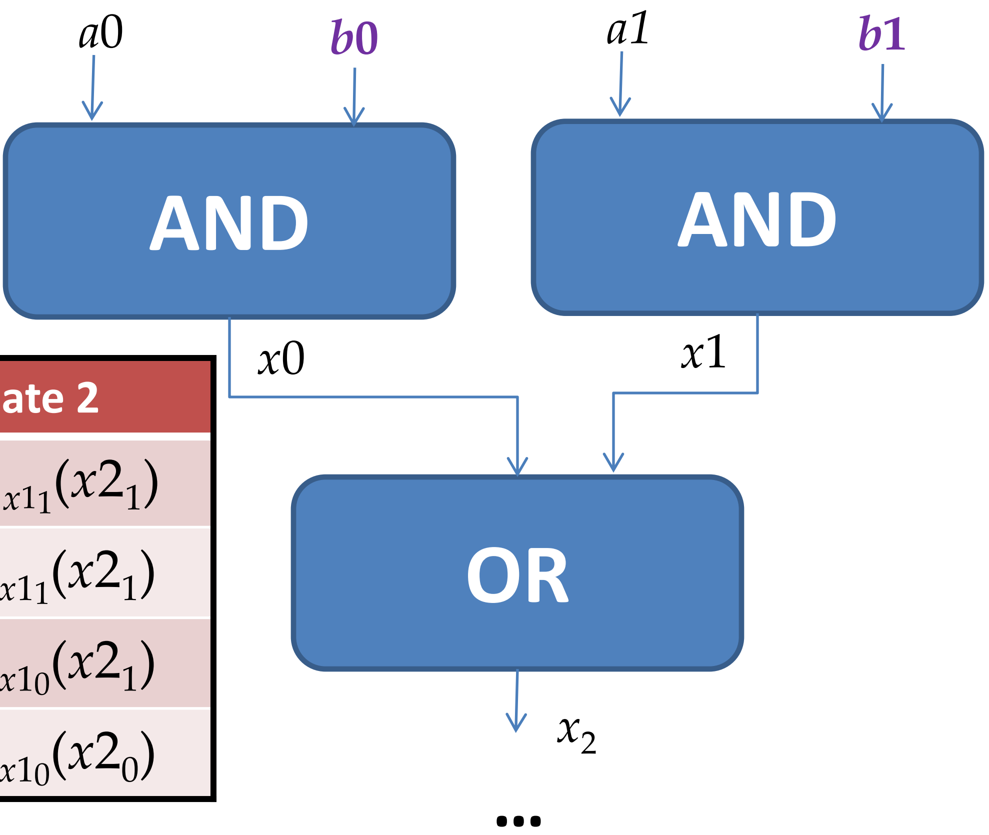
Randomly permute

Garbled And Gate
$Enc_{a0,\ b1}(x_0)$
$Enc_{a1,b1}(x_1)$
$Enc_{a1,b0}(x_0)$
$Enc_{a0,b0}(x_0)$

a_i, b_i, x_i are **random** values, chosen by the **circuit generator** but **meaningless** to the **circuit evaluator**.



Chaining Garbled Circuits



And Gate 1
$Enc_{a1_0,\ b1_1}(x1_0)$
$Enc_{a1_1,b1_1}(x1_1)$
$Enc_{a1_1,b1_0}(x1_0)$
$Enc_{a1_0,b1_0}(x1_0)$

Or Gate 2
$Enc_{x0_0,\ x1_1}(x2_1)$
$Enc_{x0_1,x1_1}(x2_1)$
$Enc_{x0_1,x1_0}(x2_1)$
$Enc_{x0_0,x1_0}(x2_0)$

Can do ***any*** computation privately this way!

Threat Model

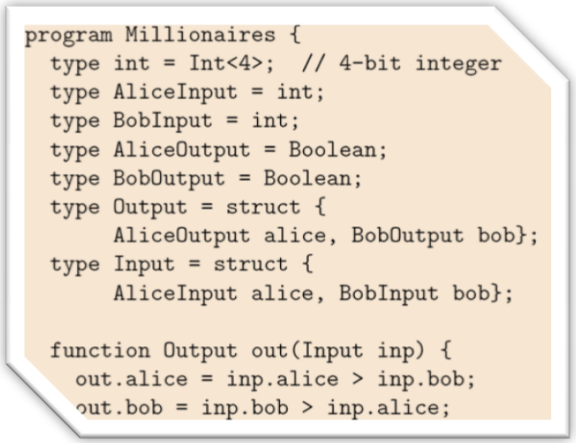
Semi-Honest (*Honest-but-Curious*) **Adversary**

Adversary follows the protocol as specified (**!**), but tries to learn more from the protocol execution transcript

May be good enough for some scenarios

We are working on efficient solutions for malicious adversaries

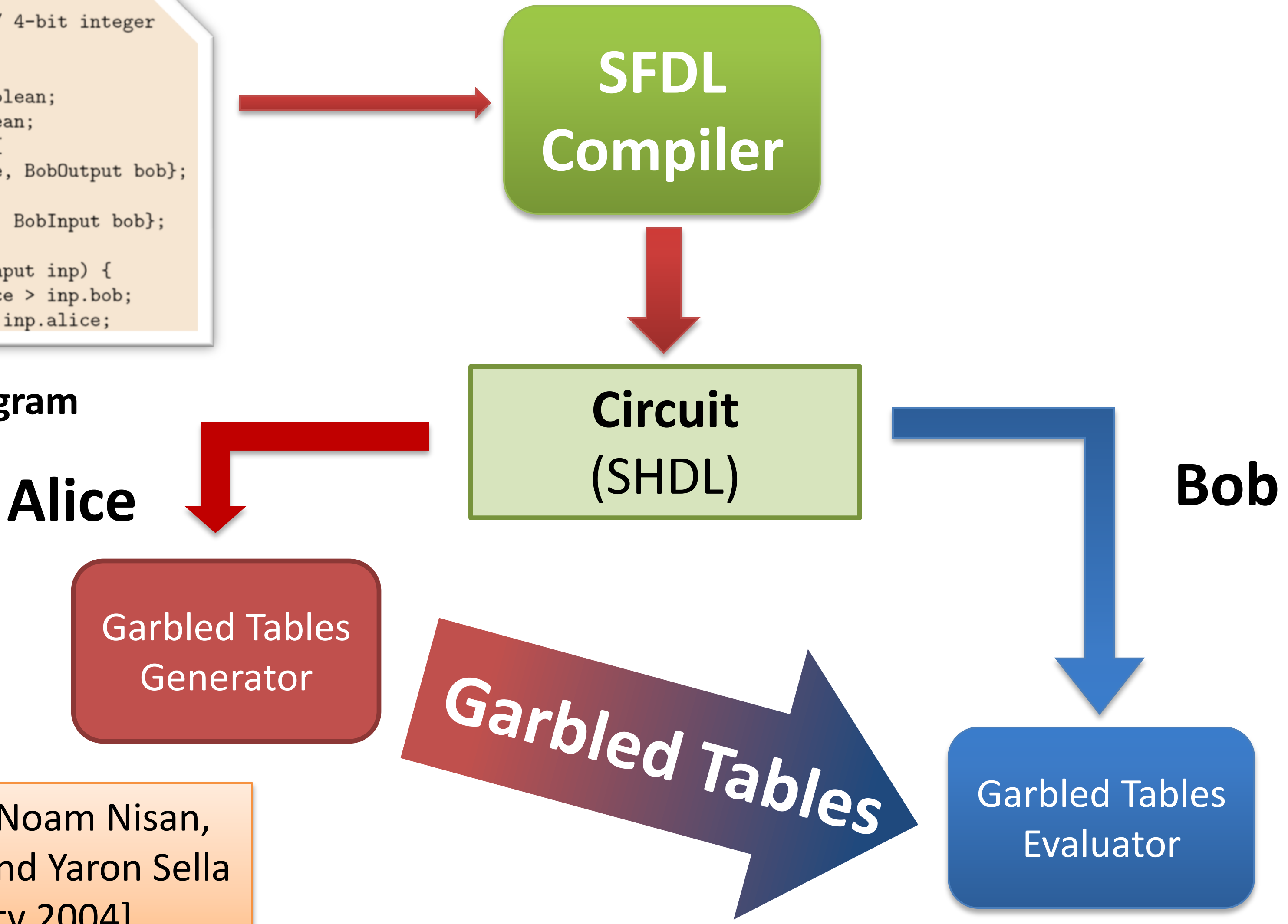
Fairplay



```
program Millionaires {
  type int = Int<4>;  // 4-bit integer
  type AliceInput = int;
  type BobInput = int;
  type AliceOutput = Boolean;
  type BobOutput = Boolean;
  type Output = struct {
       AliceOutput alice, BobOutput bob};
  type Input = struct {
       AliceInput alice, BobInput bob};

  function Output out(Input inp) {
    out.alice = inp.alice > inp.bob;
    out.bob = inp.bob > inp.alice;
```

SFDL Program



Dahlia Malkhi, Noam Nisan, Benny Pinkas and Yaron Sella [USENIX Security 2004]

Problems?

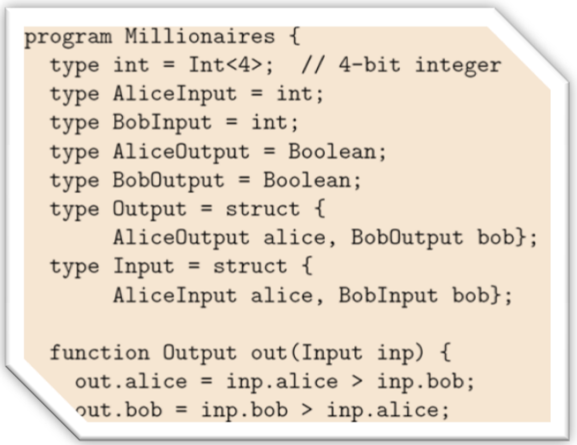
An alternative approach … would have been to apply Yao's generic secure two-party protocol…. This would have required expressing the algorithm as a circuit … and then sending and computing that circuit…. **[We] believe that the performance of our protocols is significantly better than that of applying generic protocols.**

Margarita Osadchy, Benny Pinkas, Ayman Jarrous, Boaz Moskovich. *SCiFI – A System for Secure Face Identification*. Oakland 2010.

[Generic SFE] is very fast … but the circuit size is extremely large…. Our prototype circuit compiler can compile circuits for problems of size (200, 200) but uses almost 2 GB of memory to do so…. **larger circuits would be constrained by available memory for constructing their garbled versions.**

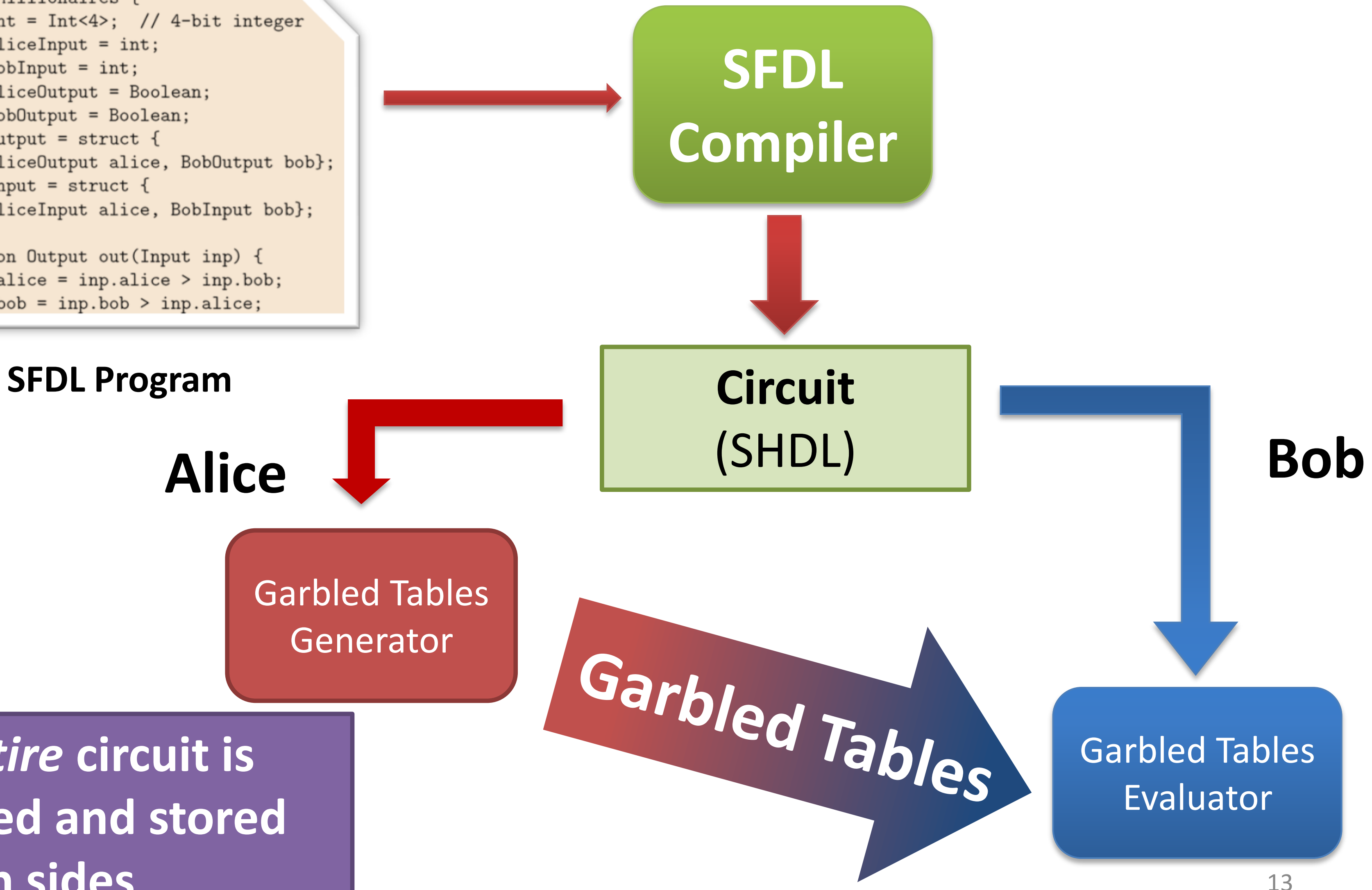
Somesh Jha, Louis Kruger, Vitaly Shmatikov. *Towards Practical Privacy for Genomic Computation*. Oakland 2008.

The Fallacy



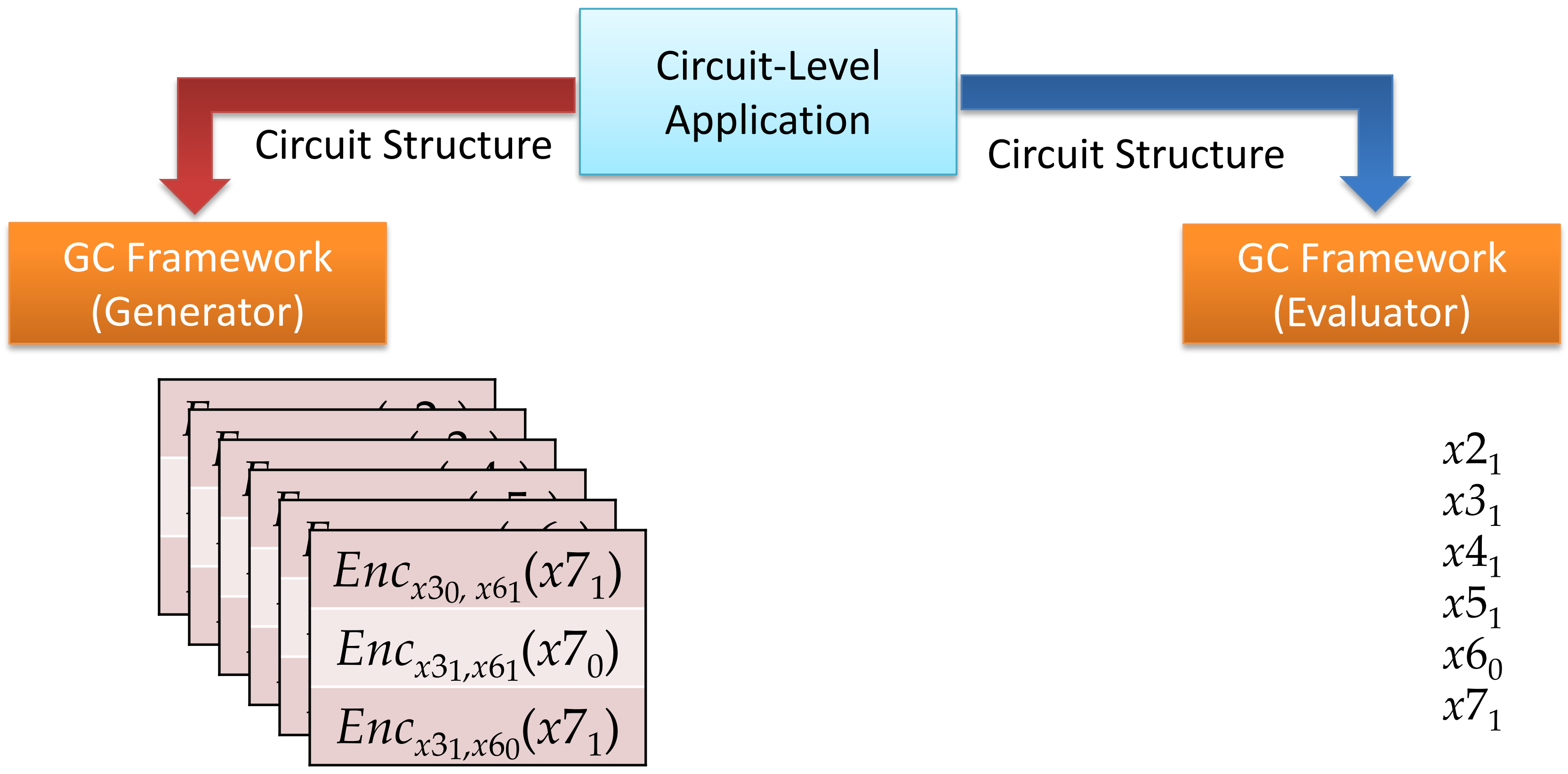
```
program Millionaires {
  type int = Int<4>;  // 4-bit integer
  type AliceInput = int;
  type BobInput = int;
  type AliceOutput = Boolean;
  type BobOutput = Boolean;
  type Output = struct {
       AliceOutput alice, BobOutput bob};
  type Input = struct {
       AliceInput alice, BobInput bob};

  function Output out(Input inp) {
    out.alice = inp.alice > inp.bob;
    out.bob = inp.bob > inp.alice;
```



The *entire* circuit is prepared and stored on both sides

Faster Garbled Circuits



Gates can be evaluated as they are generated: **pipelining**

Benefits of *Pipelining*

- Allows GC to scale to circuits of arbitrary size

We ran circuits with over a billion gates, at a rate of roughly 10 μs per gate.

- Improves the time efficiency

Problems in Existing (SFDL) Compilers

Resource-demanding SFDL compilation

It takes hours on a 40GB memory server to compile a SFDL program that implements AES.

Many optimization opportunities are missed

Circuit level

Minimize bitwidth

Reduce the number of non-free gates

Program level

Treat public and secret values differently

Example: Secure Counter

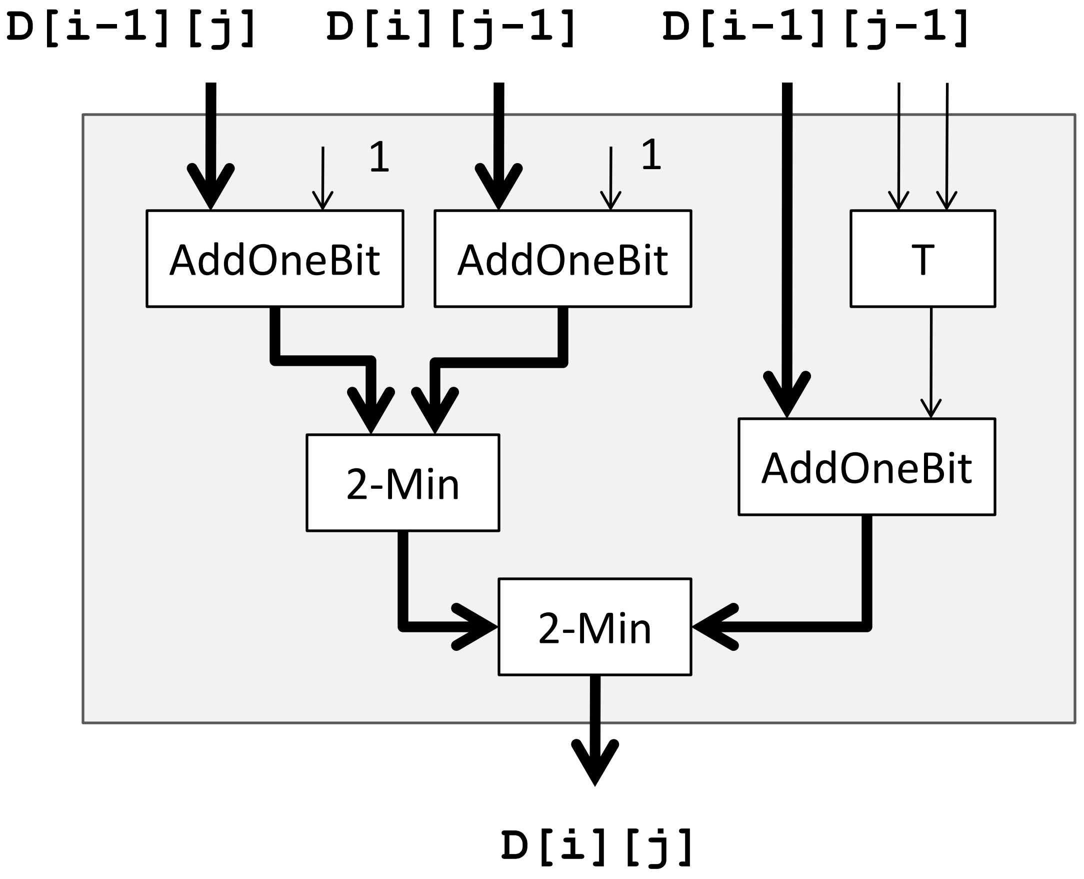
```
class Counter {
  int c = 0;
  void increment(bool b) {
    if (b) c++;
}
```

- SFDL requires pre-setting **`c`** to a fixed bit width
- For best performance, its bit width should be adjusted *dynamically*
- Saves n non-free gates (out of original $n \log n$)

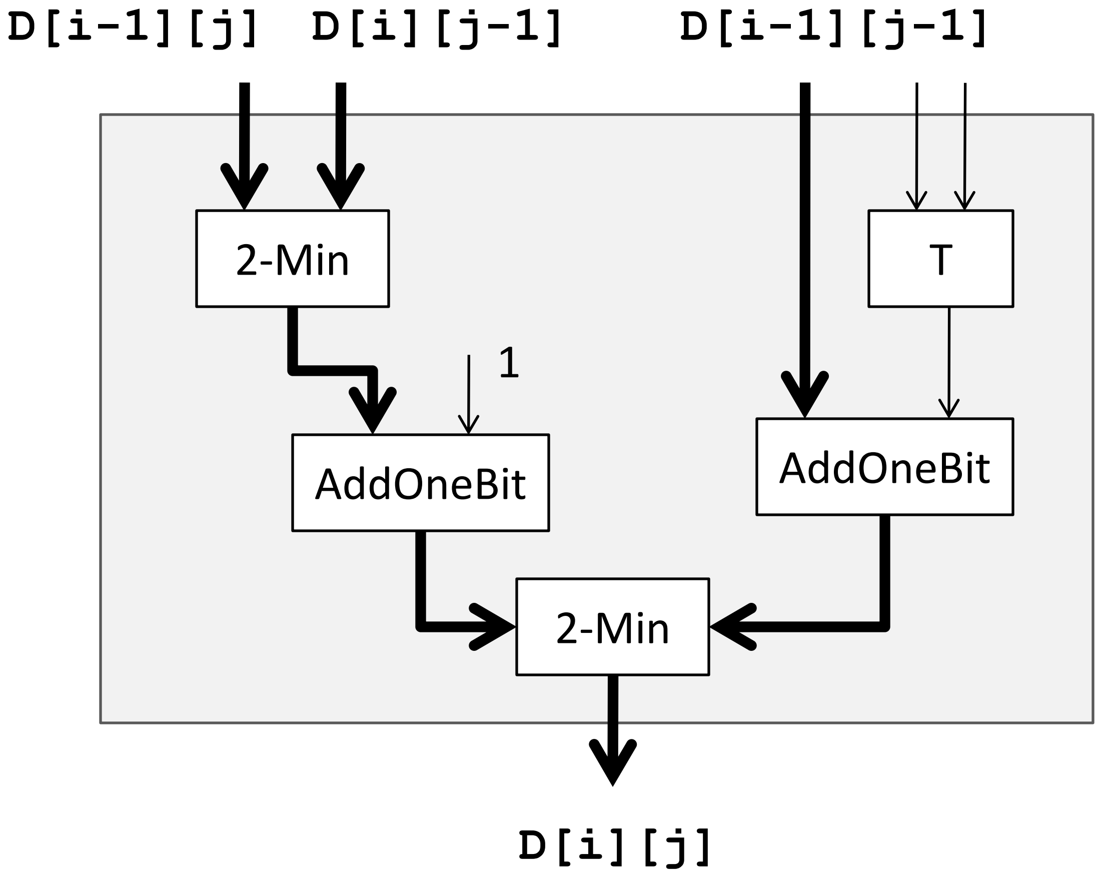
Circuit Optimization – Edit Distance

```
for (int i = 1; i < a.length; ++i)
  for (int j = 1; j < b.length; ++j) {
    T = (a[i] == b[j]) ? 0 : 1;
    D[i][j] = min(D[i-1][j]+1, D[i][j-1]+1,
                  D[i-1][j-1] + T);
  }
```

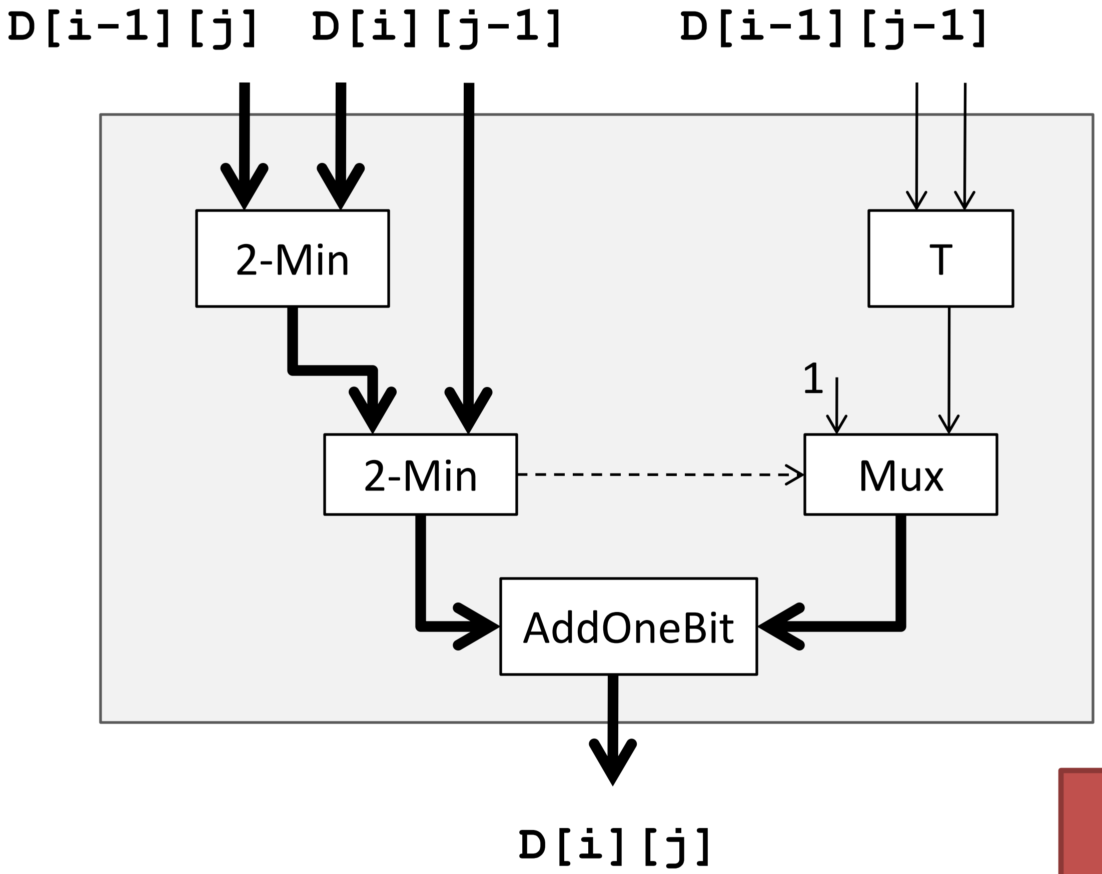
Circuit Optimization – Edit Distance



Circuit Optimization – Edit Distance

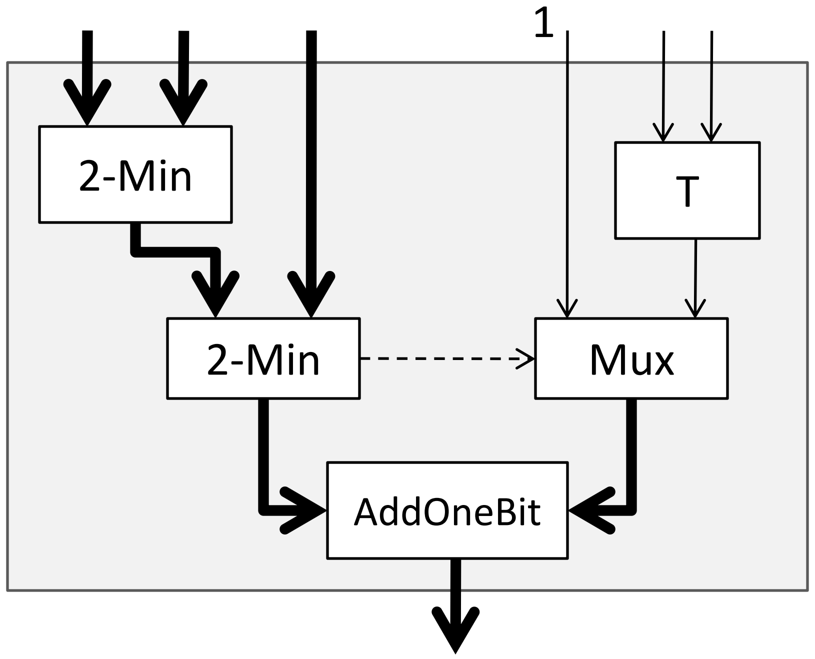


Circuit Optimization – Edit Distance



Saves about 28% of gates

Circuit Library



Through custom circuit design and the use of optimal circuit components, we strive to minimize the number of *non-free* gates

V. Kolesnikov and T. Schneider. *Improved Garbled Circuit: Free XOR Gates and Applications*. (ICALP), 2008.

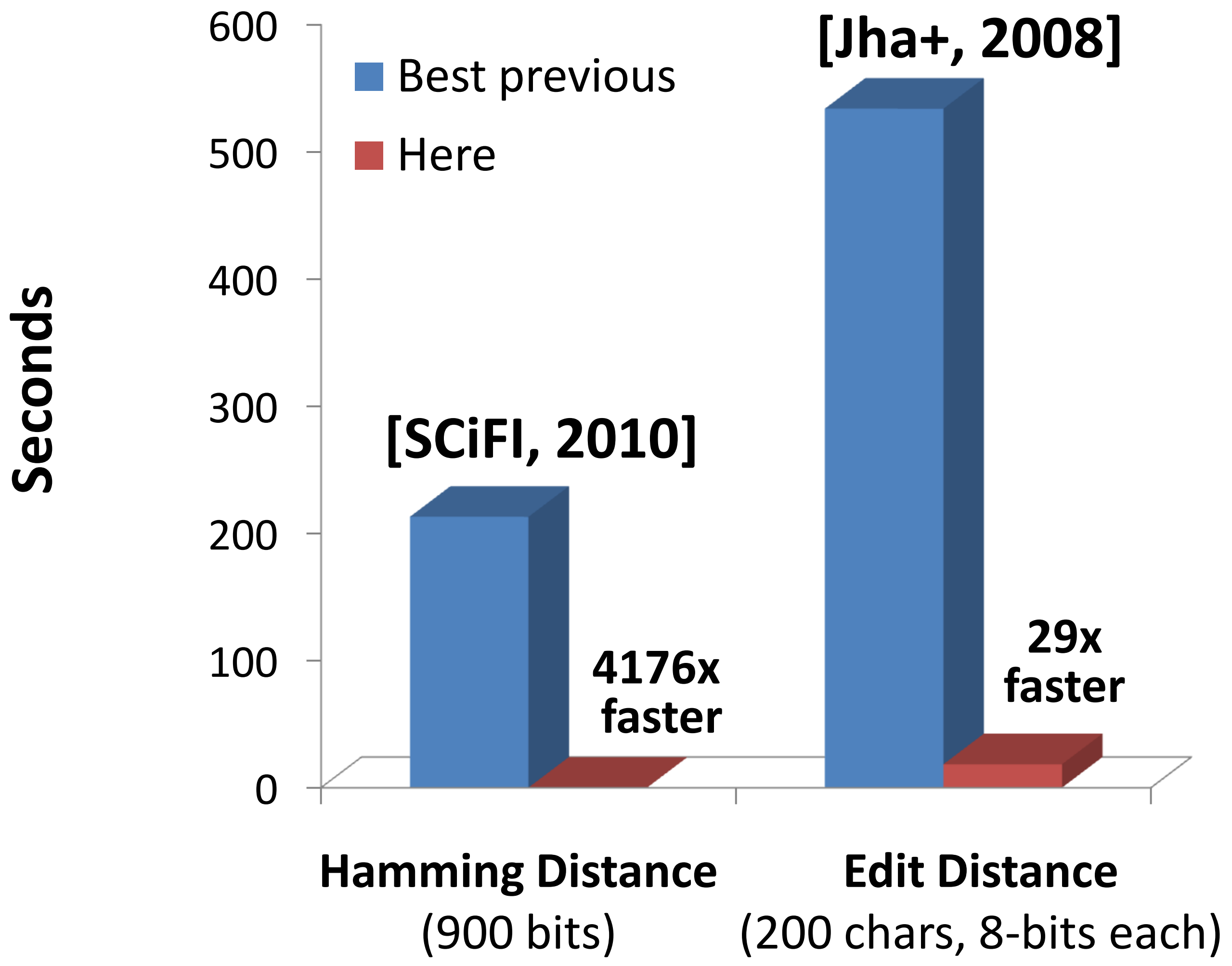
Some Results

Problem	Best Previous Result	Our Result	Speedup
Hamming Distance (Face Recognition, Genetic Dating) – two 900-bit vectors	213s [SCiFI, 2010]	**0.051s**	**4176x**
Levenshtein Distance (genome, text comparison) – two 200-character inputs	534s [Jha+, 2008]	**18.4s**	**29x**
Smith-Waterman (genome alignment) – two 60-nucleotide sequences	[Not Implementable]	**447s**	**-**
AES Encryption	3.3s [Henecka, 2010]	**0.2s**	**16.5x**

Scalable: 1 billion gates evaluated at ≈100,000 gates/second on regular PCs

Comparisons are aligned to the same security level in the semi-honest model.

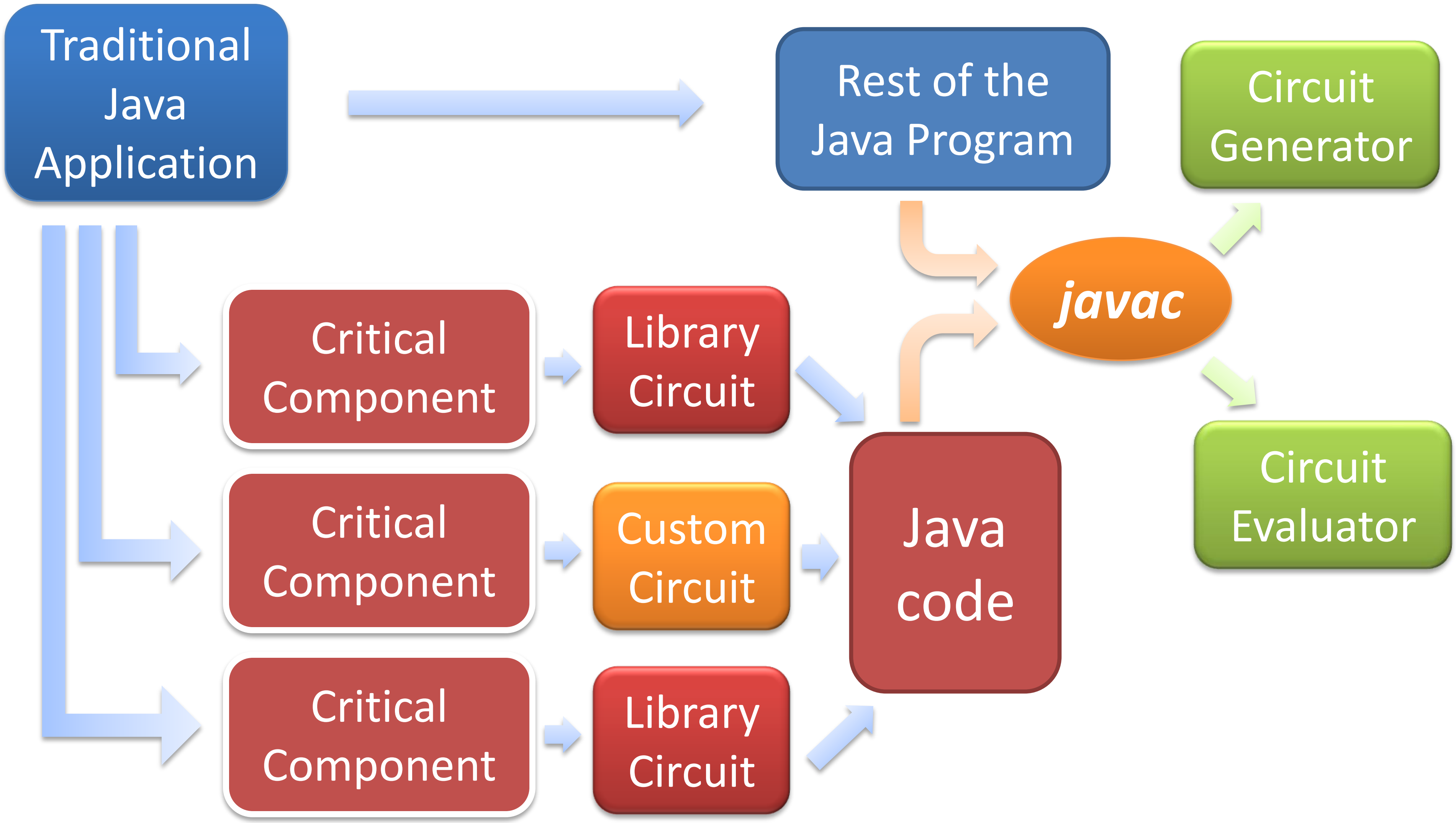
Timing Results



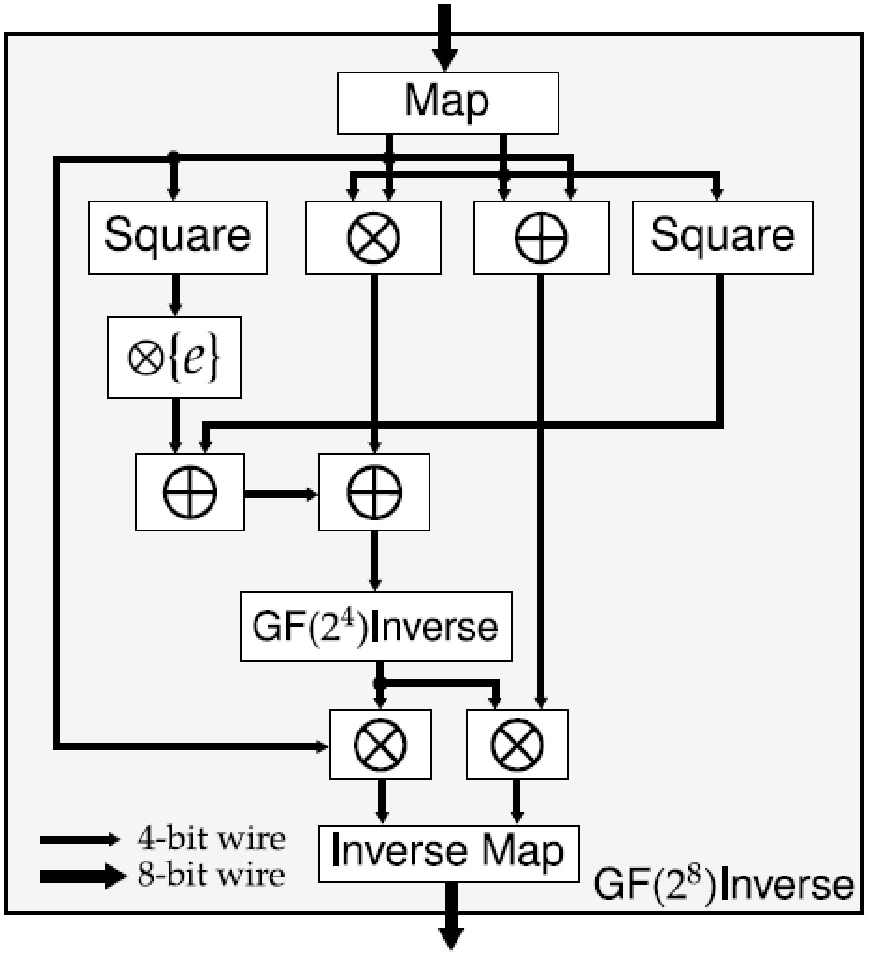
Ease of Use

- Our framework assumes no expert knowledge of cryptography
- Need basic ideas of Boolean circuits
- Circuit designs converted directly to Java programs

Use the Framework



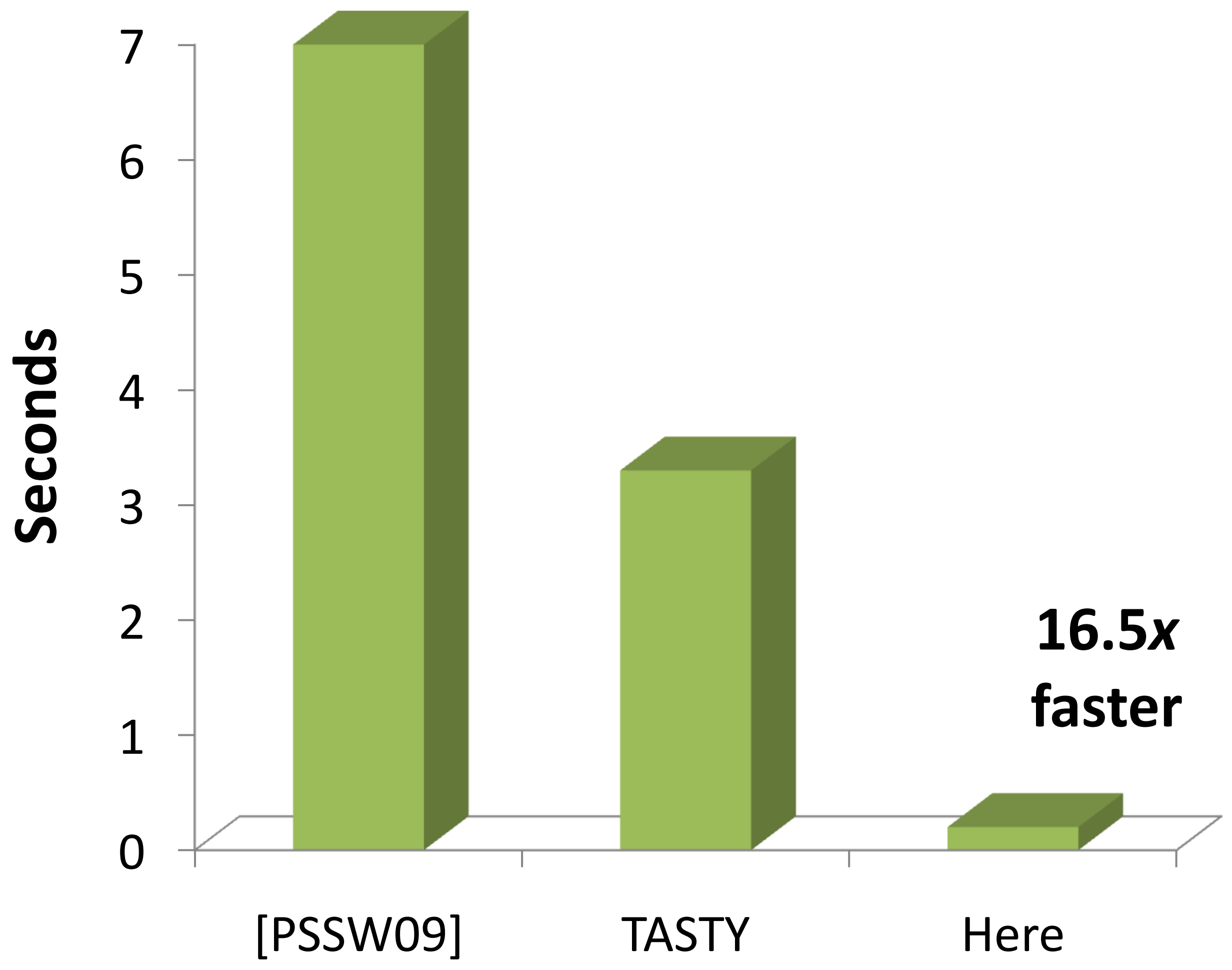
Example: AES SBox



Leveraging an existing ASIC design for AES allows us to reduce the state-of-the-art AES circuit by **30%** of non-free gates, compared to [PSSW09] and [HKSSW10]

Wolkerstorfer, et al. *An ASIC Implementation of the AES S-boxes*. RSA-CT 2002.

Time Savings: AES



[Henecka, et al. *CCS* 2010]

Conclusion

- **Pipelining** enables garbled-circuit technique to scale to large problem sizes
- **Circuit-level optimizations** can dramatically reduce performance overhead

Privacy-preserving applications can run orders of magnitude faster than previously thought.

Thanks!

Questions?

Download framework and Android demo application from **MightBeEvil.com**