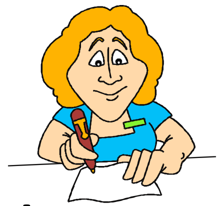
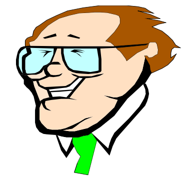
On the Performance of Private Set Intersection

Benny Pinkas, Bar-Ilan University
Thomas Schneider, TU Darmstadt
Michael Zohner, TU Darmstadt

Private Set Intersection (PSI)



Client



Server

	Client	Server
Input:	$X = x_1 \ldots x_n$	$Y = y_1 \ldots y_n$
Output:	$X \cap Y$ only	nothing

Other variants exist (e.g., both parties learn output; client learns size of intersection; compute some other function of the intersection, etc.)

Applications

- PSI is a very natural problem
  - **Matching**
    - Testing human genomes [BBC+11]
    - Proximity testing [NTL+11]
    - Relationship path discovery in social networks [MPGP09]
  - **Intersection** of suspect lists
    - Botnet detection [NMH+10]
    - Cheater detection in online games [BHLB11]

This talk

- Survey the major results
- Suggest optimizations based on new observations

- Present a new scheme

- Compare the performance of all schemes
  - On the same platform
  - Using the best optimizations that we have

A naïve PSI protocol

- A naïve solution:
  - Have A and B agree on a "cryptographic hash function" H()
  - B sends to A: $H(y_1),\ldots, H(y_n)$
  - A compares to $H(x_1),\ldots, H(x_n)$ and finds the intersection
- Does not protect B's privacy if inputs do not have considerable entropy

Preliminaries

- We set the security parameter to 128 bits
  - Namely, use symmetric and public key systems against which the best brute force attack takes 2^{128} operations.

Preliminaries

- We only consider semi-honest (passive) adversaries
  - In crypto we consider two types of adversaries:
  - Semi-honest (aka honest-but-curious) adversaries follow the protocol but try to learn more than they should
  - Malicious adversaries can behave arbitrarily
- Why discuss only semi-honest?
  - There are PSI protocols secure against malicious adversaries
  - These protocols are much less efficient
  - None of them was implemented [FNP04, JL09, HN10, CKT10, FHNP13].
  - We use OT extension…

☺ PSI secure against malicious adversaries [FHNP]

$\underline{P_1(X, m_2)}$		$\underline{P_2(Y = \{y_\alpha\}_{\alpha \in \{1 \ldots m_2\}}, m_1)}$
$1^k \longrightarrow$ $(pk, sk_1) \longleftarrow$	$\boxed{\pi_{\text{KEY}}}$	$\longleftarrow 1^k$ $\longrightarrow (pk, sk_2)$
	$\xrightarrow{E_{pk}\left(Q_1(\cdot)\right) \ldots E_{pk}\left(Q_B(\cdot)\right)}$	
$Q_1(\cdot) \ldots Q_B(\cdot) \longrightarrow$	$\boxed{\pi_{\text{POLY}}}$	$\longleftarrow E_{pk}\left(Q_1(\cdot)\right) \ldots E_{pk}\left(Q_B(\cdot)\right)$ $\longrightarrow 0/1$
Verify $sk = sk_1 + sk_2$	$\xleftarrow{sk_2}$	
		For all $\alpha \in \{1 \ldots m_2\}, j \in \{0,1\}$: $s_0^\alpha, s_1^\alpha \leftarrow_R \mathbb{M}$, $\mathcal{H}(s_j^\alpha) \to r_j \Vert \tilde{r}_j \Vert \hat{r}_j$ $q_j \stackrel{\text{def}}{=} Q_{h_j(y_\alpha)}(y_\alpha)$
	$\xleftarrow[]{e_j^\alpha = E_{pk}\left(r_j \cdot q_j + s_{1-j}^\alpha ; \hat{r}_j\right),\; t_\alpha = \tilde{r}_j \oplus y_\alpha}$	
For all $\alpha \in \{1 \ldots m_2\}, j \in \{0,1\}$: $s'_j = D_{sk}(\tilde{e}_\alpha)$, $\mathcal{H}(s'_j) \to r'_j \Vert \tilde{r}'_j \Vert \hat{r}'_j$ Check if $\exists\, x \in X\ j \in \{0,1\}$ s.t. : $t_j^\alpha = \tilde{r}_j \oplus x$, and $\tilde{e}_0^\alpha, e_1^\alpha$, consistent with $r'_1, r'_2, s'_0, s'_1, \hat{r}'_0, \hat{r}'_1$.		

Preliminaries – the random oracle model

- In the random oracle model (ROM) a specific function is modeled (in the analysis) as a random function
  - This analysis is very problematic
  - In the theory of crypto proofs in this model are considered as a heuristic
- We describe protocols that are based on the ROM
  - There are PSI protocols in the standard model [FNP04], but they are less efficient.
  - We use OT extension
    - Can be based on a non-ROM assumption
    - But a variant in the ROM is even more efficient

Public-key based Protocols

PSI based on Diffie-Hellman

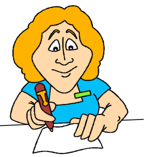
- The Decisional Diffie-Hellman assumption
  - Agree on a group G, with a generator g.
  - The assumption: for random a,b,c
    cannot distinguish (g^a, g^b, g^{ab}) from (g^a, g^b, g^c)

  - (This is a very established assumption in modern crypto)

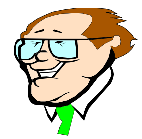
PSI based on Diffie-Hellman

- The protocol [M86, HFH99, AES03]:

α



$x_1,\ldots,x_n$



β

$y_1,\ldots,y_n$

in parallel

$(H(x_1))^\alpha,\ldots,(H(x_n))^\alpha \longrightarrow$

$\longleftarrow (H(y_1))^\beta,\ldots,(H(y_n))^\beta$

$\longleftarrow ((H(x_1))^\alpha)^\beta,\ldots,((H(x_n))^\alpha)^\beta$

$((H(y_1))^\beta)^\alpha,\ldots,((H(y_n))^\beta)^\alpha$

in parallel

Compares the two lists

(H is modeled as a random oracle. Security based on DDH)

Implementation: very simple; can be based on elliptic-curve crypto; minimal communication.

What else could we want?

PSI based on Blind RSA [CT10]

- There is also a PSI protocol based on an RSA variant

- The performance should be similar to that of DH based protocols, but
  - One party does all the hard work $\Rightarrow$ no advantage in the two parties working in parallel
  - Cannot be based on elliptic curve crypto

PSI based on Oblivious Polynomial Evaluation [FNP04] (short version)

- (Advantage: proof in the standard model)
- Implemented based on additively homomorphic encryption (Paillier, El Gamal).
- Alice generates the polynomial $P(x)=(x-x_1)(x-x_2)\cdots(x-x_n) = a_n x^n + \cdots + a_1 x + a_0$
- Alice sends homomorphic encryptions $E(a_0),E(a_1),\ldots,E(a_n)$
- $\forall y_i$ Bob uses these to evaluate and send $E(P(y_i)\cdot r_i+y_i)$
- Implementation: $O(n^2)$ exps. Can be reduced to O(nloglogn) using hashing. Too inefficient.

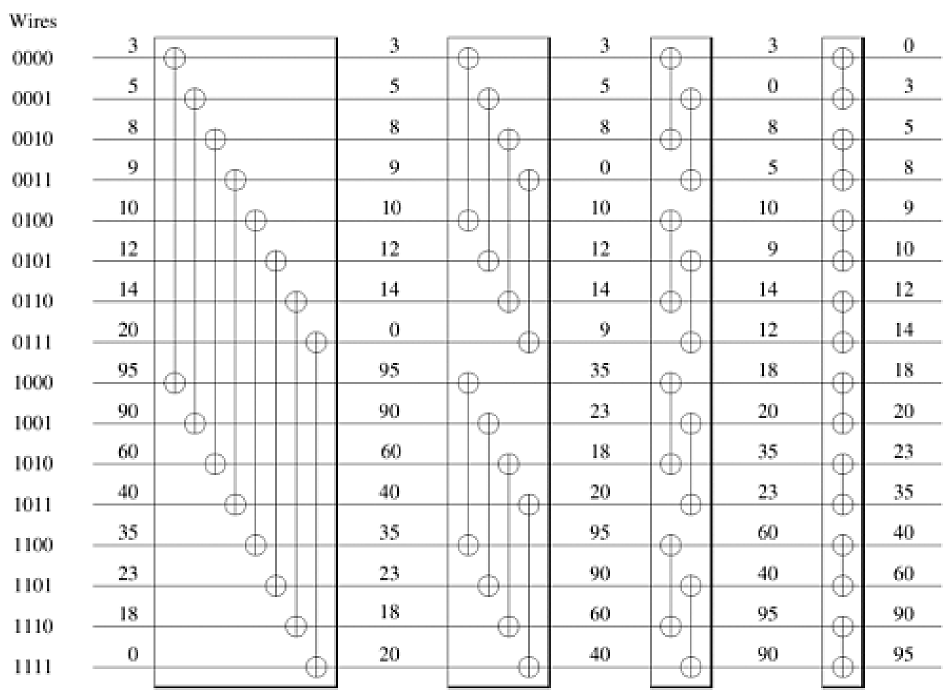
Generic Protocols

A circuit based protocol

- There are generic protocols for implementing any functionality expressed as a Binary circuit
  - GMW, Yao,…
- A naïve circuit uses n^2 comparisons of words
- Can we do better?

A circuit based protocol [HEK12]

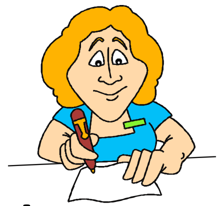
- A circuit that has three steps
  - Sort: merge two sorted lists using a bitonic merging network [Bat68]. Uses $n\log(2n)$ comparisons.



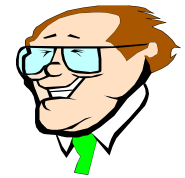
A circuit based protocol [HEK12]

- A circuit that has three steps
  - Sort: merge two sorted lists using a bitonic merging network [Bat68]. Uses $n\log(2n)$ comparisons.
  - Compare: compare adjacent items. Uses $2n$ equality checks.
  - Shuffle: Randomly shuffle results using a Waxman permutation network [W68], using $\sim n\log(n)$ swappings.
  - Overall uses $L\cdot(3n\log n + 4n)$ AND gates. (L is input length)
    - (2/3 of the AND gates are for multiplexers)

Protocols for Secure Computation are based on Oblivious Transfer (OT)



Client



Server

	Client	Server
Input:	b	X_0, X_1
Output:	X_b	nothing

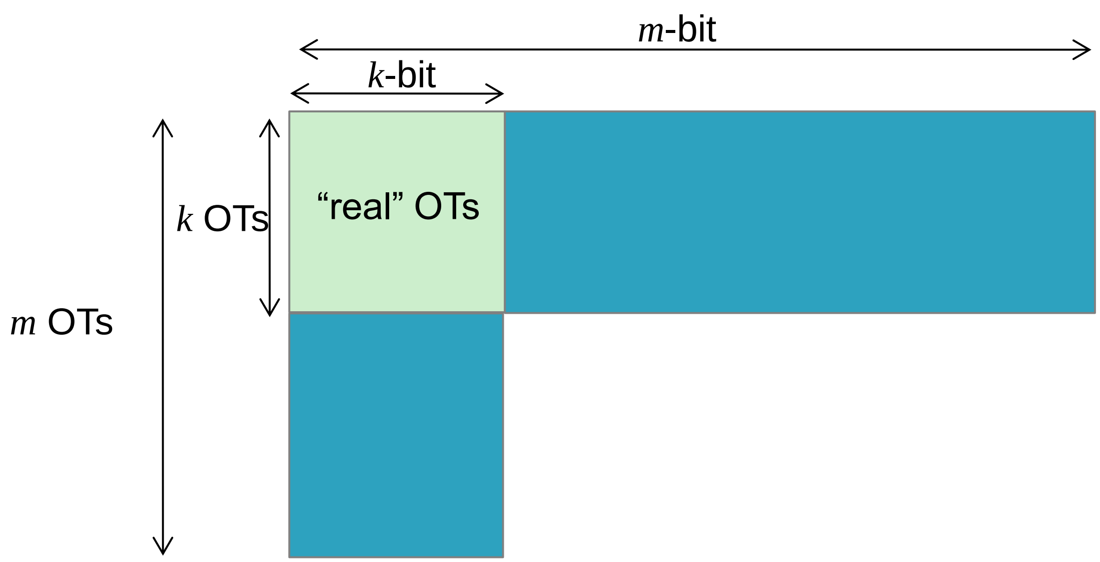
(PSI implies OT)

Side step: OT extension

- There are different OT protocols based on public-key crypto
  - - [NP01] allows ~1000 OTs per second
- [IR89] proved that there is no black-box reduction of OT to one-way functions
- OT was believed to be as (in)efficient as public-key crypto

OT extension

- OT extension runs a small number of “real” OTs, and then uses symmetric-key cryptography to compute from them many OTs [Beaver96,IKNP03]



OT extension

- Setting:
  - Bob holds *m* pairs of *l*-bit messages $(x_{i,0}, x_{i,1})$
  - Alice holds an *m*-bit string *r* and wants to obtain the value x_{i,r_i} in the *i*-th OT

- They perform *k* "real" OTs on random seeds with reverse roles (k=128 is a symmetric security parameter)
- Alice generates a random $m \times k$ bit matrix **T** and masks her choices *r*, using the seeds of the "real" OTs
- The matrix is transposed to be used for the "real" OTs

Improving OT extension [ALSZ13]

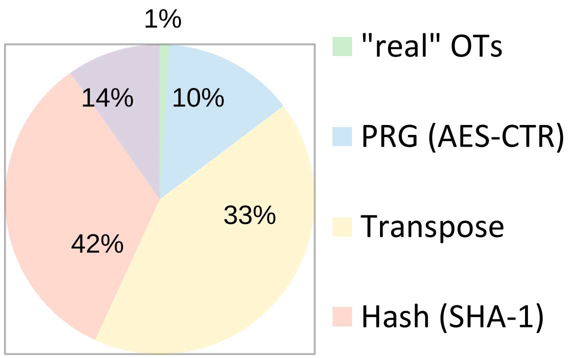
- A lot of time is spent on bit Matrix transpose – improve by using cache efficient alg
- Use parallelization

- Random OT: the two values of the sender are chosen at random
  - Cuts communication in half
  - Suitable for the GMW protocol and for our protocols!
  - Now the bottleneck is essentially the communication

Per OT:

1	# PRG evaluations	2
2	# H evaluations	1

Time distribution for 10M OTs (in 21sec):



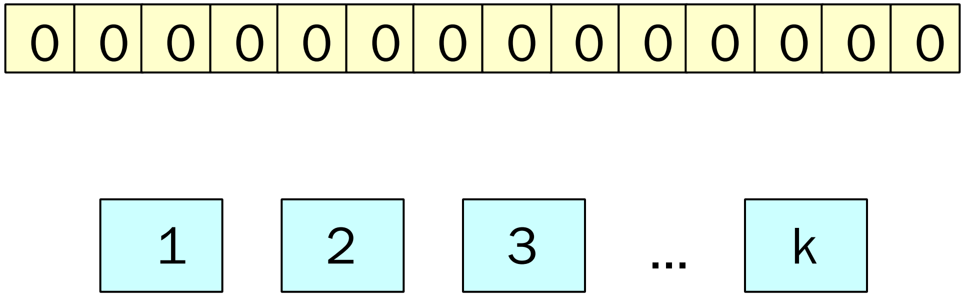
Improving the **circuit** based PSI of [HEK12]

- GMW uses two OTs per gate; Yao uses four symmetric encryptions.
  - Yao was considered much more efficient.
  - OT extension makes GMW faster than Yao.

- We noted that 2/3 of the ANDs are for multiplexers
  - A single bit chooses between two 32 bit inputs
  - For the GMW protocol, this means that instead of using 64 single-bit OTs, can use two OTs with inputs that are 32 bit long.
  - Can also base GMW on random-OT.

Garbled Bloom Filter [DCW13]

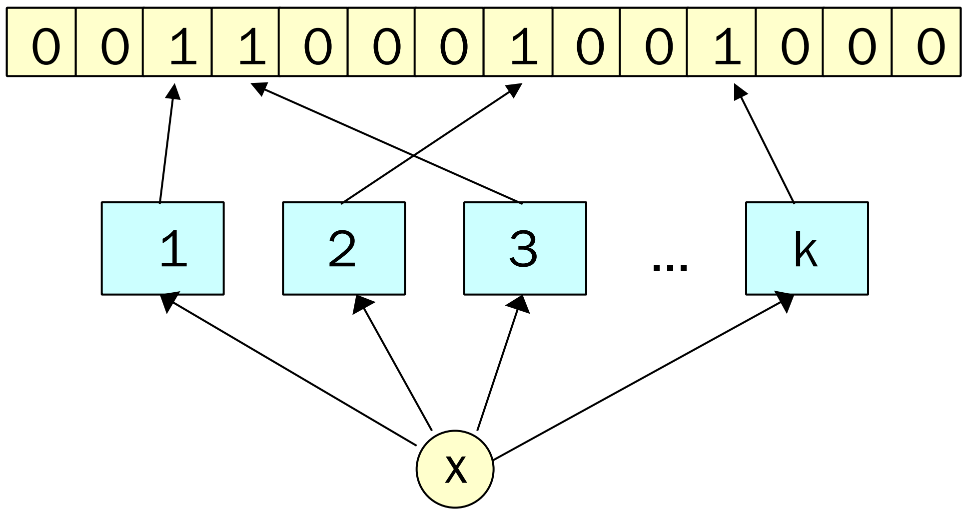
How a Bloom Filter Works



- A bit array with all zeroes initially
- k hash functions

How a Bloom Filter Works

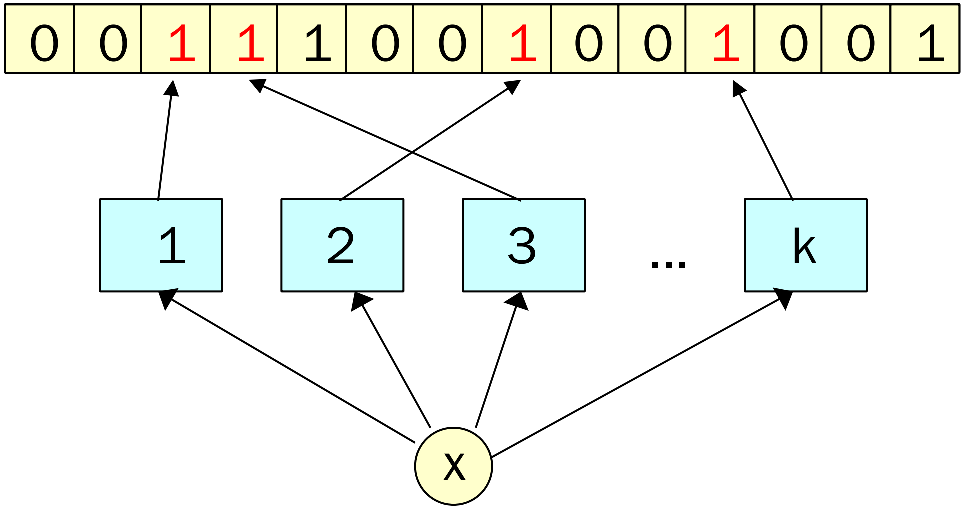
Insertion



- Hash the element using the hash functions, get k indices in the bit array
- Set the bits to 1

How a Bloom Filter Works

Lookup



- Hash the element using the hash functions
- If all corresponding bits are 1, conclude that the element is in the set

Bloom Filter Analysis

- For a false positive probability of ε, use
  - $k=1/\log\varepsilon$ hash functions
  - $m=1.44\ kn$ bits in the filter
- Must set k to be the symmetric security parameter ($k=128$ or 80).
- The resulting filter is longggg… $(184 \cdot n)$

Short story

- Use a Bloom filter where each entry is a random string
- If X is in the filter, then the XOR of the entries corresponding to X is equal to X.
- The parties run an OT per Bloom filter entry (many OTs)

(Long story) Garbled Bloom Filter [DCW13]

- Bob has items $x_1,\ldots,x_n$.
- Uses a Bloom filter where each entry *i* is a **string** G[i], s.t. the xor of the entries to which x is mapped is x.
  - $\bigoplus_{j=1\ldots k} G[h_j(x)] = x$.

- Insertion:
  - Find an index t such that $G[h_t(x)]$ is unoccupied.
  - (The failure probability is equal to the false positive prob ε)
  - Fill all other $G[h(x)]$ entries.
  - Set $G[h_t(x)]$ so that the xor of all entries is x.

(Long story) The protocol [DCW13]

- Alice generates a regular Bloom filter A[1…k].
- Bob generates a garbled Bloom filter G[1…k] (using the same hash functions)
- For each entry i in the filter, run an OT
  - Alice is the receiver, with input A[i]
  - Bob is the sender with inputs (0,G[i])
- Alice checks if x is in the intersection by checking if $\oplus_{j=1\ldots k} G[h_j(x)] = x$.
  - Alice cannot check this for values not in her input, since the probability of learning all relevant values of G[] is ε.

Our optimizations

- A complete redesign that can be implemented using *random* OT extension.

- The protocol
  - Uses much less communication
  - Becomes completely parallelizable (the original protocol required inserting items one by one)

Performance

Optimization	Party	# bits sent	# calls to H
[DCW13]	Alice	$2mk$	m
	Bob	$2m\lambda$	$2m$
[DCW13] + random OT extension of [ALSZ13]	Alice	$m\ k$	$m/2$
	Bob	$m\ \lambda$	m
Random Bloom Filter PSI **Parallelizable**	Alice	$m\ k$	$m/2$
	Bob	$n\ \lambda$	$m/2$

n/2m = 1/368

Bloom filter length:
m=1.44 · 128 · n

PSI based on OT (a new protocol)

- We first design simple protocols based on OT
- Use OT extension and hashing based constructions to the max

First step: Private equality test

- Private equality test
  - Input: Alice has x, Bob has y. Each is s bits long.
  - Output: is x=y?

Private equality test

- Alice input: 001   Bob input: 011

Private equality test

- Alice input: 001   Bob input: 011.
- Random OTs

Alice

$R_{0,0}$	
$R_{1,0}$	
	$R_{2,1}$

Bob

$R_{0,0}$	$R_{0,1}$
$R_{1,0}$	$R_{1,1}$
$R_{2,0}$	$R_{2,1}$

Private equality test

- Alice input: 001   Bob input: 011
- Random OTs

Alice

$R_{0,0}$	
$R_{1,0}$	
	$R_{2,1}$

Bob

$R_{0,0}$	$R_{0,1}$
$R_{1,0}$	$R_{1,1}$
$R_{2,0}$	$R_{2,1}$

- Bob sends $R_{0,0} \oplus R_{1,1} \oplus R_{2,1}$
- Alice computes $R_{0,0} \oplus R_{1,0} \oplus R_{2,1}$, and compares

Private set inclusion

- Input: Alice has x, Bob has $y_1,\dots,y_n$
- Output: is x in $\{y_1,\dots,y_n\}$?

- Run n Private Equality Tests in parallel.
  - Alice's OT choices for all $y_1,\dots,y_n$ are the same
  - Run only s random OTs of seeds.
  - Use a pseudo-random generator to generate from each seed n strings of length λ bits ☺
  - Send λn bits from Bob to Alice

Private set intersection

- Input: Alice has $\{x_1,\ldots,x_n\}$, Bob has $y_1,\ldots,y_n$
- Output: Intersection of $\{x_1,\ldots,x_n\}$ and $\{y_1,\ldots,y_n\}$
- Run n Private Set Inclusion protocols
- Total communication is $\underline{n^2 \lambda}$ bits
- Communication can be further reduced via hashing

Hashing

- Suppose each party uses a random hash function H(), (known to both) to hash its n items to n bins.
  - Then obviously if Alice and Bob have the same item, both of them map it to the same bin.
  - Each bin is expected to have $O(1)$ items
  - The items mapped to the bin can be compared using private equality tests, with $O(\lambda)$ communication.
  - Overall only $O(n\lambda)$ communication.
- The problem
  - Some bins have more items
  - Must hide how many items were mapped to each bin

Hashing

- Solution
  - Pad each bin with dummy items
  - so that all bins are of the size of the most populated bin

- Mapping n items to n bins
  - The expected size of a bin is $O(1)$
  - The maximum size of a bin is whp $O(\log n)$
  - Communication increases by $O(\log n)$ to be $O(n\lambda\log n)$ ☹

Hashing

- Mapping n items to about $n / \ln n$ bins
  - The expected size of a bin is $\approx O(\ln n)$
  - The maximum size of a bin is (whp) the same
  - This is ideal, since we cannot hope to pay less than the expected cost
  - Each bin has $O(\ln n)$ items. Each item can be represented by $O(\ln\ln n)$ bits.
  - The work per bin is $O(\ln n \cdot \ln\ln n)$
  - Total work is $O(n / \ln n \cdot \ln n \cdot \ln\ln n) = O(n \cdot \ln\ln n)$

Other hashing schemes

- Power of two hashing (balanced allocations)
- Cuckoo hashing

Only an asymptotic comparison was previously done

	Total #OTs	OT comm.	Overall Comm. (MB) for $n=2^{18}$
No hashing	*ns*	$n^2\lambda$	327,808
Simple hashing	3.7ns	nλ	475
Balanced hashing	2.9ns lnlnn	2nλ	939
Cuckoo hashing	(2(1+ε)n+lnn)s	(2+lnn)nλ	276

Experiments

- No previous “fair” comparison of all protocols
- We used two Intel Core2Quad desk-top PCs with 4 GB RAM, connected via Gigabit LAN
  - Inputs are 32 bit long
  - Statistical security parameter λ=40
  - Symmetric security parameter 80 or 128
  - Gigabit Ethernet

Results: run time (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	99	1224	2n PK
DH ECC	178	416	2n PK
Blind RSA	125	1982	2n PK
Circuit + GMW	807	1304	9ns logn sym
Optimized GMW	462	762	3ns logn sym
Garbled Bloom	72	154	2kn sym
Optimized G. Bloom	34	68	nk/2 sym
OT + hashing	13	14	ns/4 sym

Results: communication (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	96	192	
DH ECC	15	26	
Blind RSA	67	132	
Circuit + GMW	14760	23400	
Optimized GMW	8856	14040	
Garbled Bloom	866	1393	
Optimized G. Bloom	290	740	
OT + hashing	54	78	

DH: run time (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	99	1224	2n PK
DH ECC	178	416	2n PK
Blind RSA	125	1982	2n PK
Circuit + GMW	807	1304	9ns logn sym
Optimized GMW	462	762	3ns logn sym
Garbled Bloom	72	154	2kn sym
Optimized G. Bloom	34	68	nk/2 sym
OT + hashing	13	14	ns/4 sym

Pretty good performance!
ECC slower for 80 bit due to quality of the implementation (MIRACL vs. GIMP)

DH: communication (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	96	192	
DH ECC	15	26	
Blind RSA	67	132	
Circuit + GMW	14760	23400	
Optimized GMW	8856	14040	
Garbled Bloom	866	1393	
Optimized G. Bloom	290	740	
OT + hashing	54	78	

ECC has the best communication overhead of all protocols

Blind RSA: run time (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	99	1224	2n PK
DH ECC	178	416	2n PK
Blind RSA	125	1982	2n PK
Circuit + GMW	809	1306	9ns logn sym
Optimized GMW	465	764	3ns logn sym
Garbled Bloom	72	154	2kn sym
Optimized G. Bloom	32	66	nk/2 sym
OT + hashing	36	46	ns/4 sym

For 80 bit security, faster than a circuit (but not than DH)
Asymmetric work load between the parties

DH: communication (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	96	192	
DH ECC	15	26	
Blind RSA	67	132	
Circuit + GMW	9507	15072	
Optimized GMW	3790	5964	
Garbled Bloom	866	1393	
Optimized G. Bloom	290	740	
OT + hashing	176	276	

Circuit: run time (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	99	1224	2n PK
DH ECC	178	416	2n PK
Blind RSA	125	1982	2n PK
Circuit + GMW	807	1304	9ns logn sym
Optimized GMW	462	762	3ns logn sym
Garbled Bloom	72	154	2kn sym
Optimized G. Bloom	34	68	nk/2 sym
OT + hashing	13	14	ns/4 sym

The basic protocol is the most inefficient
Our optimizations saved more than 40% (over standard OT extension)
The result is comparable to PK based protocols
The advantage is the **generality** of a circuit based solution.

Circuit: communication (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	96	192	
DH ECC	15	26	
Blind RSA	67	132	
Circuit + GMW	14760	23400	
Optimized GMW	8856	14040	
Garbled Bloom	866	1393	
Optimized G. Bloom	290	740	
OT + hashing	54	78	

Highest communication overhead

Bloom + OT: run time (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	99	1224	2n PK
DH ECC	178	416	2n PK
Blind RSA	125	1982	2n PK
Circuit + GMW	807	1304	9ns logn sym
Optimized GMW	462	762	3ns logn sym
Garbled Bloom	72	154	2kn sym
Optimized G. Bloom	34	68	nk/2 sym
OT + hashing	13	14	ns/4 sym

The optimized Bloom protocol is 55% faster than the basic Bloom protocol
The new OT+hashing protocol even faster
Overall, OT protocols are the fastest.

Bloom + OT: run time (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	99	1224	2n PK
DH ECC	178	416	2n PK
Blind RSA	125	1982	2n PK
Circuit + GMW	807	1304	9ns logn sym
Optimized GMW	462	762	3ns logn sym
Garbled Bloom	72	154	2kn sym
Optimized G. Bloom	34	68	nk/2 sym
OT + hashing	13.5	13.8	ns/4 sym

Our OT based protocol is unaffected by the security parameter
(due to the use of symmetric crypto + communication efficiency)

Bloom + OT: communication (2^{18} items)

Protocol	80-bit	128-bit	Asymptotic
DH FFC	96	192	
DH ECC	15	26	
Blind RSA	67	132	
Circuit + GMW	14760	23400	
Optimized GMW	8856	14040	
Garbled Bloom	866	1393	
Optimized G. Bloom	290	740	
OT + hashing	54	78	

The optimized Bloom protocol reduces communication by 45%-70%.
OT protocol has the best communication, except for the ECC-DH protocol.

Using four threads (2^{18} items)

Protocol	Single thread 128-bit	Four threads	Speedup
DH FFC	1224	320	x 3.8
DH ECC	416		
Blind RSA	1982		
Circuit + GMW	1364		
Optimized GMW	762	401	x 1.9
Garbled Bloom	154		
Optimized Bloom	68	26	x 2.6
OT + hashing	14	5	x 2.8

DH and OT protocols benefit most from parallelization. Performance of circuit protocol depends more on communication.

Throughput: about a million items per 20 sec.

Communication effect on runtime (2^{16} items)

Protocol	Gigabit LAN (1000/0.2)	802.11g (54/0.2)	Intra-country (25/10)	Intra-country (10/50)	HDSPA (3.6/500)
DH ECC	104	105	108	112	116
Optimized GMW	169	1 : 2.2 371	770	1 : 2.5 1936	5311
Optimized Bloom	17	1 : 2.2 37	71	1 : 2.3 165	445
OT + hashing	3.8	1 : 1.8 5	8.8	1 : 2.6 23	78

DH is unaffected by the communication channel

OT+hashing is still the most efficient protocol.

Conclusions

- Set intersection can be efficiently applied to very large input sets
- Different settings require different protocols
  - Communication
  - Generality
  - Input lengths