A Simple Obfuscation Scheme for Pattern-Matching with Wildcards

Allison Bishop[♯] Lucas Kowalczyk[♮] Tal Malkin[♮] Valerio Pastro[♭]
Mariana Raykova[♭] Kevin Shi[♮]

♯: IEX
♮: Columbia University
♭: Yale University

August 23, 2018

Obfuscation

```
void four1(double* data, unsigned long nn)
{
    unsigned long n, mmax, m, j, istep, i;
    double wtemp, wr, wpr, wpi, wi, theta;
    double tempr, tempi;

    // reverse-binary reindexing
    n = nn<<1;
    j=1;
    for (i=1; i<n; i+=2) {
        if (j>i) {
            swap(data[j-1], data[i-1]);
            swap(data[j], data[i]);
        }
        m = nn;
        while (m>=2 && j>m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    };

    // here begins the Danielson-Lanczos section
    mmax=2;
    while (n>mmax) {
        istep = mmax<<1;
        theta = -(2*M_PI/mmax);
        wtemp = sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m=1; m < mmax; m += 2) {
            for (i=m; i <= n; i += istep) {
                j=i+mmax;
                tempr = wr*data[j-1] - wi*data[j];
                tempi = wr * data[j] + wi*data[j-1];

                data[j-1] = data[i-1] - tempr;
                data[j] = data[i] - tempi;
                data[i-1] += tempr;
                data[i] += tempi;
            }
            wtemp=wr;
            wr += wr*wpr - wi*wpi;
            wi += wi*wpr + wtemp*wpi;
        }
        mmax=istep;
    }
}
```

Obfuscation

```
void four1(double* data, unsigned long nn)
{
    unsigned long n, mmax, m, j, istep, i;
    double wtemp, wr, wpr, wpi, wi, theta;
    double tempr, tempi;

    // reverse-binary reindexing
    n = nn<<1;
    j=1;
    for (i=1; i<n; i+=2) {
        if (j>i) {
            swap(data[j-1], data[i-1]);
            swap(data[j], data[i]);
        }
        m = nn;
        while (m>=2 && j>m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    };

    // here begins the Danielson-Lanczos section
    mmax=2;
    while (n>mmax) {
        istep = mmax<<1;
        theta = -(2*M_PI/mmax);
        wtemp = sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m=1; m < mmax; m += 2) {
            for (i=m; i <= n; i += istep) {
                j=i+mmax;
                tempr = wr*data[j-1] - wi*data[j];
                tempi = wr * data[j] + wi*data[j-1];

                data[j-1] = data[i-1] - tempr;
                data[j] = data[i] - tempi;
                data[i-1] += tempr;
                data[i] += tempi;
            }
            wtemp=wr;
            wr += wr*wpr - wi*wpi;
            wi += wi*wpr + wtemp*wpi;
        }
        mmax=istep;
    }
}
```

- Proprietary algorithm?
- Cryptographic keys?

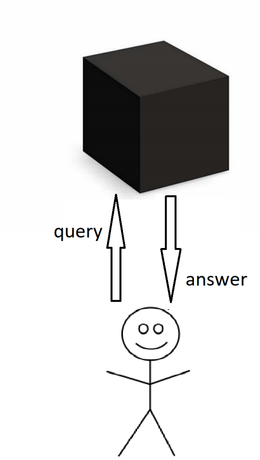
Obfuscation

```c
#include <stdio.h>

            k;double sin()
         ,cos();main(){float A=
       0,B=0,i,j,z[1760];char b[
     1760];printf("\x1b[2J");for(;;
  ){memset(b,32,1760);memset(z,0,7040)
  ;for(j=0;6.28>j;j+=0.07)for(i=0;6.28
 >i;i+=0.02){float c=sin(i),d=cos(j),e=
 sin(A),f=sin(j),g=cos(A),h=d+2,D=1/(c*
 h*e+f*g+5),l=cos      (i),m=cos(B),n=s\
in(B),t=c*h*g-f*        e;int x=40+30*D*
(l*h*m-t*n),y=            12+15*D*(l*h*n
+t*m),o=x+80*y,          N=8*((f*e-c*d*g
 )*m-c*d*e-f*g-l        *d*n);if(22>y&&
 y>0&&x>0&&80>x&&D>z[o]){z[o]=D;;;b[o]=
 ".,-~:;=!*#$@"[N>0?N:0];}}/*#****!!-*/
  printf("\x1b[H");for(k=0;1761>k;k++)
   putchar(k%80?b[k]:10);A+=0.04;B+=
     0.02;}}/*****####*******!!=;:~
       ~::==!!!**********!!!==::-
         .,~~;;;========;;;:~-.
             ..,--------,*/
```

- Proprietary algorithm?
- Cryptographic keys?

Obfuscation

```c
#include <stdio.h>

             k;double sin()
         ,cos();main(){float A=
       0,B=0,i,j,z[1760];char b[
     1760];printf("\x1b[2J");for(;;
  ){memset(b,32,1760);memset(z,0,7040)
  ;for(j=0;6.28>j;j+=0.07)for(i=0;6.28
 >i;i+=0.02){float c=sin(i),d=cos(j),e=
 sin(A),f=sin(j),g=cos(A),h=d+2,D=1/(c*
 h*e+f*g+5),l=cos      (i),m=cos(B),n=s\
in(B),t=c*h*g-f*        e;int x=40+30*D*
(l*h*m-t*n),y=            12+15*D*(l*h*n
+t*m),o=x+80*y,          N=8*((f*e-c*d*g
 )*m-c*d*e-f*g-l        *d*n);if(22>y&&
 y>0&&x>0&&80>x&&D>z[o]){z[o]=D;;;b[o]=
 ".,-~:;=!*#$@"[N>0?N:0];}}/*#****!!-*/
  printf("\x1b[H");for(k=0;1761>k;k++)
   putchar(k%80?b[k]:10);A+=0.04;B+=
     0.02;}}/*****####*******!!=;:~
       ~::==!!!**********!!!==::-
         .,~~;;;========;;;:~-.
             ..,--------,*/
```



Virtual black-box obfuscation

Virtual black-box obfuscation

Prior work

- Impossible for general circuits [BGI+01]
- Possible for limited function classes such as point functions [LPS04, Wee05] or hyperplane membership [CRV10]
- Most followup work has focused on weaker notions of obfuscation for general circuits following the construction of [GGH+13]

Virtual black-box obfuscation

Prior work

- Impossible for general circuits [BGI$^+$01]
- Possible for limited function classes such as point functions [LPS04, Wee05] or hyperplane membership [CRV10]
- Most followup work has focused on weaker notions of obfuscation for general circuits following the construction of [GGH$^+$13]

Our work

- Consider a nontrivial extension and useful to point functions
- Construct *distributional* VBB from a simple assumption

Pattern matching with wildcards

A *pattern* σ is an element $\sigma \in \{0, 1, *\}^n$
$f_\sigma(x) = 1$ if for every bit i, one of the following is true:

- $\sigma_i = x_i$
- $\sigma_i = *$

$w :=$ number of $*$'s can be a constant fraction of n

Pattern matching with wildcards

A *pattern* σ is an element $\sigma \in \{0, 1, *\}^n$
$f_\sigma(x) = 1$ if for every bit i, one of the following is true:

- $\sigma_i = x_i$
- $\sigma_i = *$

$w :=$ number of $*$'s can be a constant fraction of n

Example

$\sigma = 01 * *01$

Pattern matching with wildcards

A *pattern* σ is an element $\sigma \in \{0, 1, *\}^n$
$f_\sigma(x) = 1$ if for every bit i, one of the following is true:

- $\sigma_i = x_i$
- $\sigma_i = *$

$w :=$ number of $*$'s can be a constant fraction of n

Example

$\sigma = 01 * *01$

- $x = 010101$, $f(x) = 1$

Pattern matching with wildcards

A *pattern* σ is an element $\sigma \in \{0, 1, *\}^n$
$f_\sigma(x) = 1$ if for every bit i, one of the following is true:

- $\sigma_i = x_i$
- $\sigma_i = *$

$w :=$ number of $*$'s can be a constant fraction of n

Example

$\sigma = 01 * *01$

- $x = 010101,\ f(x) = 1$
- $x = 011001,\ f(x) = 1$

Pattern matching with wildcards

A *pattern* σ is an element $\sigma \in \{0, 1, *\}^n$
$f_\sigma(x) = 1$ if for every bit i, one of the following is true:

- $\sigma_i = x_i$
- $\sigma_i = *$

$w :=$ number of $*$'s can be a constant fraction of n

Example

$\sigma = 01 * *01$

- $x = 010101,\ f(x) = 1$
- $x = 011001,\ f(x) = 1$
- $x = 110101,\ f(x) = 0$

Pattern matching with wildcards

A *pattern* σ is an element $\sigma \in \{0, 1, *\}^n$
$f_\sigma(x) = 1$ if for every bit i, one of the following is true:

- $\sigma_i = x_i$
- $\sigma_i = *$

$w :=$ number of $*$'s can be a constant fraction of n

Applications

- Non wildcard slots in σ represent a security flaw in code. Want to check for the presence of this flaw without revealing it
- σ matches a problematic input. Want to filter out these inputs without making a user aware if he/she is otherwise unaffected

Pattern matching with wildcards

Prior work

- This function was previously studied by [BR13, BVWW16]
- From multilinear maps and from entropic LWE

Pattern matching with wildcards

Prior work

- This function was previously studied by [BR13, BVWW16]
- From multilinear maps and from entropic LWE

Our wok

- Proof of security in the generic group model
- Simple construction which relies only on elementary algebra to describe and implement

Distributional VBB for pattern matching with wildcards

Distributional VBB security

For every adversary $\mathcal{A}$ there exists a simulator S such that for every distribution $D \in \mathcal{D}_n$ and every predicate $P : \mathcal{C}_n \to \{0,1\}$:

$$\left| \Pr_{C \leftarrow \mathcal{D}_n, \mathcal{G}, \mathcal{O}^{\mathcal{G}}, \mathcal{A}}[\mathcal{A}^{\mathcal{G}}(\mathcal{O}^{\mathcal{G}}(f_\sigma, 1^n)) = P(C)] - \Pr_{C \leftarrow \mathcal{D}_n, \mathcal{S}}[\mathcal{S}^{C}(1^n) = P(C)] \right| = negl(n)$$

Distributional VBB for pattern matching with wildcards

Distributional VBB security

For every adversary $\mathcal{A}$ there exists a simulator S such that for every distribution $D \in \mathcal{D}_n$ and every predicate $P : \mathcal{C}_n \to \{0,1\}$:

$$|\Pr_{C \leftarrow \mathcal{D}_n, \mathcal{G}, \mathcal{O}^{\mathcal{G}}, \mathcal{A}}[\mathcal{A}^{\mathcal{G}}(\mathcal{O}^{\mathcal{G}}(f_\sigma, 1^n)) = P(C)] - \Pr_{C \leftarrow \mathcal{D}_n, \mathcal{S}}[\mathcal{S}^{C}(1^n) = P(C)]| = negl(n)$$

$\mathcal{O}(f_\sigma)$ where $\sigma \sim \mathcal{D}$

- Sample a random pattern σ
- Release obfuscation of f_σ

Distributional VBB for pattern matching with wildcards

Distributional VBB security

For every adversary $\mathcal{A}$ there exists a simulator S such that for every distribution $D \in \mathcal{D}_n$ and every predicate $P : \mathcal{C}_n \to \{0,1\}$:

$$\left| \Pr_{C \leftarrow \mathcal{D}_n, \mathcal{G}, \mathcal{O}^{\mathcal{G}}, \mathcal{A}} [\mathcal{A}^{\mathcal{G}}(\mathcal{O}^{\mathcal{G}}(f_\sigma, 1^n)) = P(C)] - \Pr_{C \leftarrow \mathcal{D}_n, \mathcal{S}} [\mathcal{S}^{C}(1^n) = P(C)] \right| = negl(n)$$

$\mathcal{O}(f_\sigma)$ where $\sigma \sim \mathcal{D}$

- Sample a random pattern σ
- Release obfuscation of f_σ

Simulator S

- Build 0-function simulator E
- Run $\mathcal{A}$ on E

Generic group model

Setup

- $n \times 2$ table of $2n$ "handles" in $\mathcal{H}$, where h_{ij} corresponds to $x_i = j$

	x_0	x_1	x_2	$\cdots$	x_{n-1}
0	h_{00}	h_{10}	h_{20}	$\cdots$	$h_{(n-1)0}$
1	h_{01}	h_{11}	h_{21}	$\cdots$	$h_{(n-1)1}$

Generic group model

Setup

- $n \times 2$ table of $2n$ "handles" in $\mathcal{H}$, where h_{ij} corresponds to $x_i = j$

	x_0	x_1	x_2	$\cdots$	x_{n-1}
0	h_{00}	h_{10}	h_{20}	$\cdots$	$h_{(n-1)0}$
1	h_{01}	h_{11}	h_{21}	$\cdots$	$h_{(n-1)1}$

Group oracle

- Constructs a map $\Phi : \mathcal{G} \to \mathcal{H}$
- Given $h_1, h_2 \in \mathrm{Im}\Phi$, compute $\Phi(\Phi^{-1}(h_1), \Phi^{-1}(h_2))$

Generic group model

Setup

- $n \times 2$ table of $2n$ "handles" in $\mathcal{H}$, where h_{ij} corresponds to $x_i = j$

	x_0	x_1	x_2	$\cdots$	x_{n-1}
0	h_{00}	h_{10}	h_{20}	$\cdots$	$h_{(n-1)0}$
1	h_{01}	h_{11}	h_{21}	$\cdots$	$h_{(n-1)1}$

Group oracle

- Constructs a map $\Phi : \mathcal{G} \to \mathcal{H}$
- Given $h_1, h_2 \in \mathrm{Im}\Phi$, compute $\Phi(\Phi^{-1}(h_1), \Phi^{-1}(h_2))$

Proper evaluation

- Choose $h_{0x_0}, \cdots, h_{(n-1)x_{n-1}}$ and do some math using group oracle

Proper evaluation

Handle symmetry

Given the pattern $\sigma = 01*$, the following need to behave identically:

x=010	x_0	x_1	x_2
0	$\boldsymbol{h_{00}}$	h_{10}	$\boldsymbol{h_{20}}$
1	h_{01}	$\boldsymbol{h_{11}}$	h_{21}

x=011	x_0	x_1	x_2
0	$\boldsymbol{h_{00}}$	h_{10}	h_{20}
1	h_{01}	$\boldsymbol{h_{11}}$	$\boldsymbol{h_{21}}$

Polynomial interpolation

Setup

- Sample and fix a degree-n polynomial $p \in \mathbb{Z}_p[x]$ such that $p(0) = 0$
- $a_1, \cdots, a_n \sim \mathbb{Z}_p$ and $f(x) = a_1 x + \cdots + a_n x^n$

Polynomial interpolation

Setup

- Sample and fix a degree-n polynomial $p \in \mathbb{Z}_p[x]$ such that $p(0) = 0$
- $a_1, \cdots, a_n \sim \mathbb{Z}_p$ and $f(x) = a_1 x + \cdots + a_n x^n$

Handle distribution

- $\sigma_i \neq j : \tilde{h}_{ij}$ is random in $\mathbb{Z}_p$

Example for $\sigma = 01*$

	x_0	x_1	x_2
0		r	
1	r		

Polynomial interpolation

Setup

- Sample and fix a degree-n polynomial $p \in \mathbb{Z}_p[x]$ such that $p(0) = 0$
- $a_1, \cdots, a_n \sim \mathbb{Z}_p$ and $f(x) = a_1 x + \cdots + a_n x^n$

Handle distribution

- $\sigma_i \neq j : \tilde{h}_{ij}$ is random in $\mathbb{Z}_p$
- $\sigma_i = j : \tilde{h}_{ij} = p(2i + j)$

Example for $\sigma = 01*$

	x_0	x_1	x_2
0	$p(0)$	r	
1	r	$p(3)$	

Polynomial interpolation

Setup

- Sample and fix a degree-n polynomial $p \in \mathbb{Z}_p[x]$ such that $p(0) = 0$
- $a_1, \cdots, a_n \sim \mathbb{Z}_p$ and $f(x) = a_1 x + \cdots + a_n x^n$

Handle distribution

- $\sigma_i \neq j$: $\tilde{h}_{ij}$ is random in $\mathbb{Z}_p$
- $\sigma_i = j : \tilde{h}_{ij} = p(2i + j)$
- $\sigma_i = * : \tilde{h}_{ij} = p(2i + j) \, \forall j$

Example for $\sigma = 01*$

	x_0	x_1	x_2
0	$p(0)$	r	$p(4)$
1	r	$p(3)$	$p(5)$

Function evaluation

Function evaluation

- Pick the samples $\{\tilde{h}_{ix_i}\}_{i=0}^{n-1}$
- Constructing interpolating polynomial $\hat{p}$
- Output 1 if $\hat{p}(0) = 0$

Attacks in the clear

Error-correction for Reed-Solomon codes

- Treat the table of $2n$ handles as $2n$ samples of a degree-n polynomial with some number of errors $e = n - w$
- Berlekamp-Welch algorithm can decode if $w > \frac{n}{2}$

Attacks in the clear

Error-correction for Reed-Solomon codes

- Treat the table of $2n$ handles as $2n$ samples of a degree-n polynomial with some number of errors $e = n - w$
- Berlekamp-Welch algorithm can decode if $w > \frac{n}{2}$

Observations

- Attacks require nonlinear computations over input-output pairs
- Correct evaluation of $\hat{p}(0)$ only requires a linear computation

Construction (in the exponent)

Setup

- Sample and fix a degree-n polynomial $p \in \mathbb{Z}_p[x]$ such that $p(0) = 0$
- Fix a cyclic group $\mathcal{G}$ with generator g and prime order p

Construction (in the exponent)

Setup

- Sample and fix a degree-n polynomial $p \in \mathbb{Z}_p[x]$ such that $p(0) = 0$
- Fix a cyclic group $\mathcal{G}$ with generator g and prime order p

Handle distribution

- $\sigma_i \neq j : h_{ij}$ is random in $\mathcal{G}$
- $\sigma_i = j : h_{ij} = g^{p(2i+j)}$
- $\sigma_i = * : h_{ij} = g^{p(2i+j)} \forall j$

Example for $\sigma = 01*$

	x_0	x_1	x_2
0	$g^{p(0)}$	r	$g^{p(4)}$
1	r	$g^{p(3)}$	$g^{p(5)}$

Polynomial interpolation in the exponent

Function evaluation

- $p(x) = \sum_{i=0}^{n-1} y_i b_i(x)$: Lagrange interpolating polynomial over $\{(x_i, y_i)\}$

Polynomial interpolation in the exponent

Function evaluation

- $p(x) = \sum_{i=0}^{n-1} y_i b_i(x)$: Lagrange interpolating polynomial over $\{(x_i, y_i)\}$
- Compute Lagrange coefficients $C_i := b_i(0) = \prod_{j \neq i} \frac{-2j - x_j}{2i - x_i - x_j + 2j}$

Polynomial interpolation in the exponent

Function evaluation

- $p(x) = \sum_{i=0}^{n-1} y_i b_i(x)$: Lagrange interpolating polynomial over $\{(x_i, y_i)\}$
- Compute Lagrange coefficients $C_i := b_i(0) = \prod_{j \neq i} \frac{-2j - x_j}{2i - x_i - x_j + 2j}$
- Compute $\prod_{i=0}^{n-1} h_{ix_i}^{C_i}$

Polynomial interpolation in the exponent

Function evaluation

- $p(x) = \sum_{i=0}^{n-1} y_i b_i(x)$: Lagrange interpolating polynomial over $\{(x_i, y_i)\}$
- Compute Lagrange coefficients $C_i := b_i(0) = \prod_{j \neq i} \frac{-2j-x_j}{2i-x_i-x_j+2j}$
- Compute $\prod_{i=0}^{n-1} h_{ix_i}^{C_i}$

Correctness

- If each $h_{ix_i} = g^{p(2i+x_i)}$, then $\prod_{i=0}^{n-1} h_{ix_i}^{C_i} = g^{\sum_{i=1}^{n} p(2i+x_i)C_i} = g^{p(0)}$
- If any h_{ix_i} is a random group element, then output is random

Generic group simulators

Generic group simulators

Internal group representation

- S: $\mathcal{G}$

Example element

- $g^{p(3)}$

Generic group simulators

Internal group representation

- S: $\mathcal{G}$
- E: $(\mathbb{Z}_p[\mathbf{c}_1, \cdots, \mathbf{c}_{2n}], +)$

Example element

- $g^{p(3)}$
- $\mathbf{c}_{11}$

Generic group simulators

Internal group representation

- S: $\mathcal{G}$
- E: $(\mathbb{Z}_p[\mathbf{c}_1, \cdots, \mathbf{c}_{2n}], +)$
- M: $(\mathbb{Z}_p[\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}], +)$

Example element

- $g^{p(3)}$
- $\mathbf{c}_{11}$
- $3\mathbf{a}_1 + 9\mathbf{a}_2$

Generic group simulators

Internal group representation

- S: $\mathcal{G}$
- E: $(\mathbb{Z}_p[\mathbf{c}_1, \cdots, \mathbf{c}_{2n}], +)$
- M: $(\mathbb{Z}_p[\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}], +)$

Example element

- $g^{p(3)}$
- $\mathbf{c}_{11}$
- $3\mathbf{a}_1 + 9\mathbf{a}_2$
- $\mathbf{b}_1$

Security game

Things to keep track of in generic group model

- Correspondence between handles and internal group elements
- When two different generic group simulators differ

Security game

Things to keep track of in generic group model

- Correspondence between handles and internal group elements
- When two different generic group simulators differ

Definition (Simultaneous oracle game)

An adversary is given access to a pair of oracles $(\mathcal{G}_M, \mathcal{G}_*)$, where $\mathcal{G}_*$ is $\mathcal{G}_M$ with probability $1/2$ and $\mathcal{G}_S$ with probability $1/2$. In each round, the adversary asks the same query to both oracles. The adversary wins the game if he guesses correctly the identity of $\mathcal{G}_*$.

Simultaneous oracle game between S and M

Simultaneous oracle game between S and M

Definition (Evaluation map in the exponent)

Given fixed values $a_1, \cdots, a_n, b_1, \cdots, b_{n-w}$, we have the evaluation map

$$\begin{aligned} \phi : \mathbb{Z}[\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}] &\longrightarrow \mathcal{G} \\ F(\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}) &\longmapsto g^{F(a_1, \cdots, a_n, b_1, \cdots, b_{n-w})} \end{aligned}$$

Simultaneous oracle game between S and M

Definition (Evaluation map in the exponent)

Given fixed values $a_1, \cdots, a_n, b_1, \cdots, b_{n-w}$, we have the evaluation map

$$\begin{aligned} \phi : \mathbb{Z}[\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}] &\longrightarrow \mathcal{G} \\ F(\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}) &\longmapsto g^{F(a_1, \cdots, a_n, b_1, \cdots, b_{n-w})} \end{aligned}$$

Notation

- $\mathcal{H}_S^t, \mathcal{H}_M^t$ — the set of handles returned by the simulator up to round t

Simultaneous oracle game between S and M

Definition (Evaluation map in the exponent)

Given fixed values $a_1, \cdots, a_n, b_1, \cdots, b_{n-w}$, we have the evaluation map

$$\begin{aligned} \phi : \mathbb{Z}[\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}] &\longrightarrow \mathcal{G} \\ F(\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}) &\longmapsto g^{F(a_1, \cdots, a_n, b_1, \cdots, b_{n-w})} \end{aligned}$$

Notation

- $\mathcal{H}_S^t, \mathcal{H}_M^t$ — the set of handles returned by the simulator up to round t
- $\Psi : \mathcal{H}_M^t \to \mathcal{H}_S^t$ — the adversary's identification of handles returned by each simulator when given the same query

Simultaneous oracle game between S and M

Definition (Evaluation map in the exponent)

Given fixed values $a_1, \cdots, a_n, b_1, \cdots, b_{n-w}$, we have the evaluation map

$$\begin{aligned} \phi : \mathbb{Z}[\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}] &\longrightarrow \mathcal{G} \\ F(\mathbf{a}_1, \cdots, \mathbf{a}_n, \mathbf{b}_1, \cdots, \mathbf{b}_{n-w}) &\longmapsto g^{F(a_1, \cdots, a_n, b_1, \cdots, b_{n-w})} \end{aligned}$$

Notation

- $\mathcal{H}_S^t, \mathcal{H}_M^t$ — the set of handles returned by the simulator up to round t
- $\Psi : \mathcal{H}_M^t \to \mathcal{H}_S^t$ — the adversary's identification of handles returned by each simulator when given the same query
- $\Phi_M : \mathbb{Z}[\mathbf{a}, \mathbf{b}] \to \mathcal{H}_M, \Phi_S : \mathcal{G} \to \mathcal{H}_S$ — each simulator's internal mapping of group elements to handles

Inductive hypothesis

Suppose the adversary has made t queries so far and has $\mathcal{H}_S^t, \mathcal{H}_M^t$ satisfying the following:

Inductive hypothesis

Suppose the adversary has made t queries so far and has $\mathcal{H}_S^t, \mathcal{H}_M^t$ satisfying the following:

1. For each round $i \leq t$ and query answers h_i^s, h_i^m, either $\Psi(h_i^m) = h_i^s$ or both $h_i^s \notin \mathcal{H}_S^{i-1}$ and $h_i^m \notin \mathcal{H}_M^{i-1}$

Inductive hypothesis

Suppose the adversary has made t queries so far and has $\mathcal{H}_S^t, \mathcal{H}_M^t$ satisfying the following:

1. For each round $i \leq t$ and query answers h_i^s, h_i^m, either $\Psi(h_i^m) = h_i^s$ or both $h_i^s \notin \mathcal{H}_S^{i-1}$ and $h_i^m \notin \mathcal{H}_M^{i-1}$
2. For every $h^s \in \mathcal{H}_S^t$, $\exists! f \in \mathbb{Z}_p[\mathbf{a}, \mathbf{b}]$ such that $\Phi_S \circ \phi(f) = i_S(h^s)$ and $\Psi^{-1}(h^s) = \Phi_M(f)$

Visualization of (2)

$$\begin{tikzcd}
\mathbb{Z}_p[\mathbf{a},\mathbf{b}] \arrow[r, "\Phi_M{,}\simeq"] \arrow[d, two heads, "\phi"'] & \mathrm{Im}(\Phi_M) & \mathcal{H}_M^t \arrow[l, hook', "i_M"'] \arrow[d, "\Psi{,}="] \\
G \arrow[r, "\Phi_S{,}\simeq"'] & \mathrm{Im}(\Phi_S) & \mathcal{H}_S^t \arrow[l, hook', "i_S"] \arrow[ull, "\exists!"']
\end{tikzcd}$$

The failure event

Given t rounds of simulation, on round $t+1$:

The failure event

Given t rounds of simulation, on round $t+1$:

1. Adversary performs the query $h^1 \cdot h^2$ to Simulator M and $\Psi(h^1) \cdot \Psi(h^2)$ to Simulator S

The failure event

Given t rounds of simulation, on round $t+1$:

1. Adversary performs the query $h^1 \cdot h^2$ to Simulator M and $\Psi(h^1) \cdot \Psi(h^2)$ to Simulator S
2. Simulator M returns h^m and Simulator S returns h^s

The failure event

Given t rounds of simulation, on round $t+1$:

1. Adversary performs the query $h^1 \cdot h^2$ to Simulator M and $\Psi(h^1) \cdot \Psi(h^2)$ to Simulator S
2. Simulator M returns h^m and Simulator S returns h^s
3. The inductive hypothesis holds for $t+1$ unless $h^m \notin \mathcal{H}_M^t$ but $h^s \in \mathcal{H}_S^t$

The failure event

Given t rounds of simulation, on round $t+1$:

1. Adversary performs the query $h^1 \cdot h^2$ to Simulator M and $\Psi(h^1) \cdot \Psi(h^2)$ to Simulator S
2. Simulator M returns h^m and Simulator S returns h^s
3. The inductive hypothesis holds for $t+1$ unless $h^m \notin \mathcal{H}_M^t$ but $h^s \in \mathcal{H}_S^t$

- $h^m = \Phi_M(f_m)$ for some f_m. By the inductive hypothesis $\exists!\, f_s$ such that $\Phi_S \circ \phi(f_s) = i_S(h^s)$

The failure event

Given t rounds of simulation, on round $t+1$:

1. Adversary performs the query $h^1 \cdot h^2$ to Simulator M and $\Psi(h^1) \cdot \Psi(h^2)$ to Simulator S
2. Simulator M returns h^m and Simulator S returns h^s
3. The inductive hypothesis holds for $t+1$ unless $h^m \notin \mathcal{H}_M^t$ but $h^s \in \mathcal{H}_S^t$

- $h^m = \Phi_M(f_m)$ for some f_m. By the inductive hypothesis $\exists!\, f_s$ such that $\Phi_S \circ \phi(f_s) = i_S(h^s)$
- Failure event is $f_s - f_m \in \ker \phi$ but $f_s - f_m$ is nontrivial

The failure event

Given t rounds of simulation, on round $t+1$:

1. Adversary performs the query $h^1 \cdot h^2$ to Simulator M and $\Psi(h^1) \cdot \Psi(h^2)$ to Simulator S
2. Simulator M returns h^m and Simulator S returns h^s
3. The inductive hypothesis holds for $t+1$ unless $h^m \notin \mathcal{H}_M^t$ but $h^s \in \mathcal{H}_S^t$

- $h^m = \Phi_M(f_m)$ for some f_m. By the inductive hypothesis $\exists!\, f_s$ such that $\Phi_S \circ \phi(f_s) = i_S(h^s)$
- Failure event is $f_s - f_m \in \ker\phi$ but $f_s - f_m$ is nontrivial
- This is just a combinatorial probability calculation

Conclusion

Conclusion

- We give obfuscation scheme for pattern matching with wildcards from a simpler generic group assumption

Conclusion

- We give obfuscation scheme for pattern matching with wildcards from a simpler generic group assumption
- The construction itself is simple to describe and implement in any standard group library

Conclusion

- We give obfuscation scheme for pattern matching with wildcards from a simpler generic group assumption
- The construction itself is simple to describe and implement in any standard group library
- We give a new framework for formalizing generic group proofs via the simultaneous oracle game

Conclusion

- We give obfuscation scheme for pattern matching with wildcards from a simpler generic group assumption
- The construction itself is simple to describe and implement in any standard group library
- We give a new framework for formalizing generic group proofs via the simultaneous oracle game

Thanks for listening!

Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang.
On the (im)possibility of obfuscating programs.
In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 1–18, 2001.

Zvika Brakerski and Guy N. Rothblum.
Obfuscating conjunctions.
In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 416–434, 2013.

Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs.
Obfuscating conjunctions under entropic ring LWE.
In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 147–156, 2016.

Ran Canetti, Guy N. Rothblum, and Mayank Varia.
Obfuscation of hyperplane membership.
In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 72–89, 2010.

Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters.
Candidate indistinguishability obfuscation and functional encryption for all circuits.
In *FOCS*, 2013.

Ben Lynn, Manoj Prabhakaran, and Amit Sahai.
Positive results and techniques for obfuscation.
In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 20–39, 2004.

Hoeteck Wee.
On obfuscating point functions.
In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 523–532, 2005.