Multi-Inputs Non-Interactive Functional Encryption (MINI-FE) without Trusted Authorities and Applications to Private Grading

Vincenzo Iovino

Team Fictional Encryption

January 15, 2021

Based on the work: https://eprint.iacr.org/2015/1119

Motivation

- Alice, Bob and Eve are the Judges for an online Hackaton and Eve and Peggy are the candidates who submit some projects to be evaluated.
  - Each Judge submits a grade for each candidate, e.g., a number that is 0 (reject the project), 1 (borderline), 2 (accept) and a candidate is evaluated with the average of the grades submitted by all Judges.
- For privacy reasons Alice, Bob and Eve do not want to submit their grades in clear.
  - Also Eve and Peggy may be wish that the grades are not public to not generate conflicts of interest with the Judges.
- We do not want any trusted party and setup assumptions beyond the fact that the identities (e.g., public-keys) of both Judges and Candidates are public.

Failure of FE or e-voting

- The problem seems very similar to e-voting.
  - Unfortunately, in traditional e-voting there are trusted election authorities that colluding together can always leak the votes.
- In Multi-Inputs Functional Encryption, a function of N ciphertexts encrypting the grades can be computed.
  - Unfortunately, Functional Encryption also needs one or more trusted authorities that colluding together can leak the grades of the Judges.

The MINI-FE setting

- There are n participants, and each of them generates a pair of public- and secret-keys. Each participant knows the public-keys of all other participants.
- The i-th participant can encode an input X_i with her/his own secret-key and the public-keys of the others to generate a ciphertext CT_i.
- There is a public Evaluation function that takes all ciphertexts and compute $f(x_1, \ldots, x_n)$ where f is a given function (we explain later the functions we achieve).
- MINI-FE can be seen as a sort of Multi-Inputs Functional Encryption but *without* central/trusted authorities.

Syntax

MINI-FE

1. $\mathsf{Setup}(1^\lambda)$, on input the security parameter in unary, outputs *public* parameters pp.
2. $\mathsf{KeyGen}(\mathsf{pp})$, on input the public parameters pp outputs a *public-key* Pk and a *secret-key* Sk.
3. $\mathsf{Encode}(\mathsf{pp}, j, \mathsf{id}, \mathsf{Sk}, (\mathsf{Pk})_{i\in[n]-\{j\}}, v)$, on input the public parameters pp, the secret-key Sk of j, the identifier $\mathsf{id} \in \{0,1\}^\lambda$ of the ceremony, the public keys $(\mathsf{Pk}_i)_{i\in[v]-\{j\}}$ of the other Judges, and a grade $v \in D$, outputs CT;
4. $\mathsf{VerifyCT}(\mathsf{pp}, \mathsf{Pk}, \mathsf{id}, \mathsf{CT})$, on input the public parameters pp, a public-key Pk of a Judge, the identifier $\mathsf{id} \in \{0,1\}^\lambda$, and a ciphertext CT, outputs a value in $\{\perp, \mathsf{OK}\}$;
5. $\mathsf{Eval}(\mathsf{pp}, \mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathsf{id}, \mathsf{CT}_1, \ldots, \mathsf{CT}_n)$, on input the public parameters pp, the public-keys of all Judges, the identifier $\mathsf{id} \in \{0,1\}^\lambda$, and the ciphertexts submitted by all Judges, outputs $y \in \Sigma \cup \{\perp\}$.

Privacy

- Setup phase. $\mathcal{C}$ generates $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, choose a random bit $b \leftarrow \{0,1\}$ and runs $\mathcal{A}_0$ on input pp;
- Corruption phase. $\mathcal{A}_0$, on input pp, outputs a set $S \subset [n]$ of indices of Judges it wants to corrupt.
- Key Generation Phase. For all $i \in [n]$ the challenger generates n pairs $(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and runs $\mathcal{A}_1^{\mathsf{Grade}(\cdot)}$ on input $(\mathsf{Pk}_i, \mathsf{Sk}_i)_{i \in S}$ and $(\mathsf{Pk}_i)_{i \in [n]-S}$.
- Query phase. The adversary $\mathcal{A}_1$ has access to a stateful oracle Grade that on input an identifier $\mathsf{id} \in \{0,1\}^\lambda$ and a pair of vectors $\vec{v}_0 \triangleq (v_{0,1}, \ldots, v_{0,n})$ and $\vec{v}_1 \triangleq (v_{1,1}, \ldots, v_{1,n})$ outputs the ciphertexts $(\mathsf{Encode}(\mathsf{pp}, 1, \mathsf{id}, \mathsf{Sk}_1, (\mathsf{Pk}_i)_{i \in [n]-\{1\}}, v_{b,1}), \ldots, \mathsf{Encode}(\mathsf{pp}, n, \mathsf{id}, \mathsf{Sk}_n, (\mathsf{Pk}_i)_{i \in [n]-\{n\}}, v_{b,$
- Output. At some point the adversary outputs its guess b'.
- Winning condition. The adversary wins the game if the following conditions hold:
  - $b' = b$; $v_{0,i} = v_{1,i}$ for any $i \in S$; S has cardinality $< n$, $\vec{v}_0$ and $\vec{v}_1$ are vectors of n values in D and $\mathsf{id} \in \{0,1\}^\lambda$.
  - for any $(\vec{v}_0, \vec{v}_1)$ for which $\mathcal{A}$ asked a query to Grade it holds that: for any vector $\vec{v}$, $F(\vec{v}_0') = F(\vec{v}_1')$ where for $b = 0, 1$ $\vec{v}_b'$ is the vector equal to $\vec{v}$ in all indices in S and equal to $\vec{v}_b$ elsewhere.

Verifiability

- A MINI-FE scheme should be verifiable.
- This means that the Evaluation procedure should be able to detect if a Judge submitted an invalid grade/decision.
- For example a Candidate could corrupt a Judge and asks him/her to submit a grade higher than the maximum in an attempt to increase the average. Without verifiability this attack would go undetected.
- The schemes we construct are verifiable.

Definition of Verifiability (very technical)

Definition of verifiability

For all pp $\leftarrow$ Setup(1^λ), $j \in [n]$, Pk, CT, there exists a vote $v \in D$ such that: for all identifiers id $\in \{0,1\}^\lambda$, all except negligible fraction of $(\mathsf{Pk}_i, \mathsf{Sk}_i)_{i \in [n]-\{j\}}$ such that for all $i \in [n] - \{j\}$ $(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and all $v_i \in D$ with $i \in [n] - \{j\}$ and all except negligible fraction of $(\mathsf{CT}_i)_{i \in [n]-\{j\}}$ satisfying $\mathsf{CT}_i \leftarrow \mathsf{Encode}(\mathsf{pp}, i, \mathsf{id}, \mathsf{Sk}, (\mathsf{Pk})_{j \in [n]-\{i\}}, v_i)$, it holds that $\mathsf{Eval}(\mathsf{pp}, \mathsf{Pk}_1, \ldots, \mathsf{Pk}_{j-1}, \mathsf{Pk}, \mathsf{Pk}_{j+1}, \ldots, \mathsf{Pk}_n, \mathsf{id}, \mathsf{CT}_1, \ldots, \mathsf{CT}_{j-1}, \mathsf{CT}, \mathsf{CT}_{j+1}, \ldots, \mathsf{CT}_n)$ outputs either $F(v_1, \ldots, v_{j-1}, v, v_{j+1}, \ldots, v_n)$ or $\perp$

(continued...)

For all pp $\leftarrow$ Setup(1^λ), $j \in [n]$, Pk, CT, if $\mathsf{VerifyCT}(\mathsf{pp}, \mathsf{Pk}, \mathsf{id}, \mathsf{CT}) = \mathsf{OK}$ then: for all identifiers id $\in \{0,1\}^\lambda$, all except negligible fraction of $(\mathsf{Pk}_i, \mathsf{Sk}_i)_{i \in [n]-\{j\}}$ such that for all $i \in [n] - \{j\}$ $(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, all $v_1, \ldots, v_{n_1} \in D$, all except negligible fraction of $(\mathsf{CT}_i)_{i \in [n]-\{j\}}$ such that for all $i \in [v] - \{j\}$ $\mathsf{CT}_i \leftarrow \mathsf{Encode}(\mathsf{pp}, j, \mathsf{id}, \mathsf{Sk}, (\mathsf{Pk})_{i \in [n]-\{j\}}, v)$, it holds that $\mathsf{Eval}(\mathsf{pp}, \mathsf{Pk}_1, \ldots, \mathsf{Pk}_{j-1}, \mathsf{Pk}, \mathsf{Pk}_{j+1}, \ldots, \mathsf{Pk}_n, \mathsf{id}, \mathsf{CT}_1, \ldots, \mathsf{CT}_{j-1}, \mathsf{CT}, \mathsf{CT}_{j+1}, \ldots, \mathsf{CT}_n) \neq \perp$

Overview of Our Main scheme

Our main scheme (simplified)

- Use bilinear maps (for simplicity henceforth we use symmetric groups) and pair any element with Hash(id) and evaluate in the target group.
- In ceremony id Judge j submits her grade v_j as

  $\mathsf{CT}_j \triangleq \mathbf{e}(g^{y_j}, \mathsf{Hash}(\mathsf{id}))^{x_j} \cdot \mathbf{e}(g^{v_j}, \mathsf{Hash}(\mathsf{id}))$, where g^{y_j} is computed from the PKs g^{x_j}'s in the following way:

  - $g^{y_j} \triangleq g^{\sum_{k<j} x_k - \sum_{k>j} x_k}$
  - Note that $\prod_{j\in[n]} g^{x_j y_j} = 1$.
- If $g_{\mathsf{id}} \triangleq \mathbf{e}(g, \mathsf{Hash}(\mathsf{id}))$ then $\mathsf{CT}_j = g_{\mathsf{id}}^{v_j} g_{\mathsf{id}}^{x_j y_j} \rightarrow \prod_{j\in[n]} \mathsf{CT}_j = g_{\mathsf{id}}^{\sum v_j}$.
- The sum of v_j's can be computed by brute force and the *average* of the grades $(\sum_{j\in[n]} v_j)/n$ can be computed.
- In the target group we construct hash function creating new generators for each ceremony in such a way that the PK for any participant in the new generator can be calculated by the other participants and the SKs stay unchanged.

NIZK proofs

Adding proofs of well-formedness

- Consider the pair (g, g^{y_i}) as El Gamal PKs and the pair $(g^{x_i}, g^{y_i x_i} g^{v_i})$ as El Gamal encryption with randomness x_i, public-key g^{y_i} and plaintext v_i.
- The Cramer *et al.*'s (CDS) sigma protocol can prove that v_i is either 0 or 1 without revealing which one.
- Our work identical except that g is in target group.
- Variable becomes $g \triangleq \mathbf{e}(g', \mathrm{Hash}(\mathrm{id}))$ where g' is in bilinear group and Hash function mapping the input to the base group.
- Straightforward to verify that CDS work when g has this form.

Beyond Average Grading

Dead or Alive

- In *Dead or Alive decisions* (or Accept/Reject) the evaluation procedure has to compute the predicate $P_{\neq 0}$ that is true iff at least one Judge accepted the project of the candidate.
- Change average grade ceremonies so that if the j-th Judge submits 0, she sets $v_j = 0$, otherwise she sets v_j to *random*, i.e, $\mathsf{CT}_j \triangleq \mathbf{e}(g^{y_j}, \mathsf{Hash}(\mathsf{id}, \mathcal{I}))^{x_j} \cdot \mathbf{e}(g^{v_j}, \mathsf{Hash}(\mathsf{id}, \mathcal{I}))$, but with v_j set as described before.

Unanimity

- Similar except that we invert the setting of v_j by choosing $v_j = 0$ if the Judge submits 1 or choosing v_j at random for 0

Security reduction

Goal

Reduce to BDDH, e.g. hard to distinguish $\mathbf{e}(g,g)^{abc}$ from random given g, g^a, g^b, g^c.

Strategy

- For any query we consider the two challenge vectors $\vec{x}_0, \vec{x}_1$, e.g. $\vec{x}_0 \triangleq 00101, \vec{x}_1 \triangleq 10010$.
- In any iteration identify two positions i, j in which $\vec{x}_0$ and $\vec{x}_1$ have symmetric difference, e.g. $\mathbf{0}0\mathbf{1}01$ and $\mathbf{1}0\mathbf{0}10$.
- Swap them, e..g $\vec{z} \triangleq 10001$; now $\vec{z}$ is more similar to $\vec{x}_1$.
- continue to swap until $\vec{z} = \vec{x}_1$.
- In any iteration plant the challenge $\mathbf{e}(g,g)^{abc}$ of the BDDH in the two positions i, j.
- Setting: $\mathsf{Hash}(\mathsf{id}^\star) = g^c, \mathsf{Pk}_i = g^b, \mathsf{Pk}_j = g^c$.

Our Implementations

- We implemented MINI-FE routines for both average grading, dead or alive and unanimity ceremonies in a shared library libminife.so
- + we implemented a framework that makes possible to design paring-based cryptosystems and protocols in a way that is compatible both with CiFEr library and the Stanford pbc library.
- + a demo for evaluating candidates in an online Hackaton.
- All the code and documentation can be found here: https://github.com/cryptohackathon/MINI-FE

Compatibility with Stanford pbc library

Example

```
#include <stdio.h>
#include "pairings.h"
int main(void){
element_t a,b,a2,b2,y,T,T4,_T4; // all elements are of type element_t
pairing_t p; // pairing instance
pairing_init_set_str(p,Param); // Param is a static global constant
element_init_G1(a,p); // a is an element of G1 - all the following elements are
     associated to the pairing instance p
element_init_G1(a2,p);
element_init_G2(b,p); element_init_G2(b2,p); // b and b2 are in G2
element_random(a); element_random(b); // choose random elements in the group
     where a,b have been initialized
element_mul(a2,a,a); // a2=a^2
element_mul(b2,b,b); // b2=b^2
element_init_GT(T,p); // T belongs to the target group
element_init_GT(T4,p); element_init_GT(_T4,p);
element_pairing(T,a,b); //T=e(a,b)
element_pairing(T4,a2,b2); //T4=e(a2,b2)=e(a^2,b^2)=e(a,b)^4=T^4
element_init_Zr(y,p); // y belongs to Zr with r order of the groups
element_set1(y); // y=1
element_add(y,y,y); element_add(y,y,y);// y=y+y+=4
element_pow_zn(_T4,T,y); // _T4=T^4
printf("%d\n",element_cmp(T4,_T4)); // test if T4 == _T4 - should output 0
return 0;
}
```

Future directions

MINI-FE over blockchains

- Independently, we are implementing a “proxy” that makes transparent for secure multy-party protocols to communicate over a blockchain.
- The protocol over TCP is changed just replacing the IP/port addresses of the remote parties by localhost/ports of the proxy running on the local machine and the proxy handles in a transparent way the communication among parties.
- Advantages: All messages are logged into the blockchain and you can restart the computation from where you started in case of energy interruption.
- This also allows to implement a perfect *broadcast* channel needed by MINI-FE protocols.