Secure Function Evaluation in Mobile Environments

Benjamin Mood and Kevin Butler University of Oregon

Cellphones are prevalent and smartphones are increasingly common. Phones are increasingly being used for common tasks such as browsing the web, email, and e-commerce. However, privacy mechanisms for phones are still developing. Consider if someone was conducting an auction on their phone. They want to prevent the identity of the person who won the auction from being known. The solution is secure function evaluation.

Secure Function Evaluation

Secure function evaluation (SFE) is the cryptographic means to compute a function with two or more people's input while preserving privacy by not revealing data to any other party.

The most common implementation of SFE is garbled circuits. Any program can be represented as a garbled circuit. Garbled circuits are a way of hiding the output by masking the wires for the gates for the virtual circuit. Instead of one or zero, the wires are renamed by using the SHA-1 hash function to mask what they contain. However, using the SHA-1 hash function is computationally intensive. This is a particular problem for mobile processors due to their computing and battery constraints.

We used an implementation of garbled circuits called Fairplay. The Fairplay compiler provides a means for generating and evaluating garbled circuits. Fairplay takes a text file in the Secure Function Definition Language (SFDL) and compiles it into a circuit which Fairplay can then run. The same program is run by both parties involved.

SFDL example: Billionaires program

```
/*
 * Check which of two Billionaires is richer
 */
program Billionaires {
      type int = Int<32>; // 32-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 output(Input input) {
            output.alice = (input.alice > input.bob);
            output.bob = (input.bob > input.alice);
      }
}
```

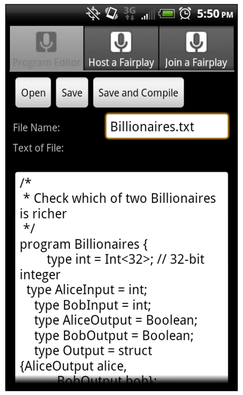
Sample runs

If these values were entered

Alice	Bob
AliceInput = 1000	BobInput = 1001
Output.alice: 0	Output.bob: 1

Alice	Bob
AliceInput = 1001	BobInput = 1000
Output.alice: 1	Output.bob: 0

Alice	Bob
AliceInput = 1000	BobInput = 1000
Output.alice: 0	Output.bob: 0



FairplayAN

We developed FairplayAN, an Android application created from the original Java code of Fairplay. We ported the code to Android and modified the input and output interfaces to assume operational usability of the code.

Optimization

We observed that generating and evaluating circuits was slow compared to a PC. We profiled the code and saw there were parts of the program which were computationally intense. Notably, one of these areas was evaluating SHA-1. We deployed a SHA-1 function written in C in an attempt to circumvent inefficiencies from the Java middleware.

Recent phone processors have vector coprocessors capable of single instruction multiple data (SIMD) operations. We made the app SIMD capable. The SIMD instruction set used by Android phones is called NEON, which allows for up to four precalculations done at the same time; this parallelism reduces the time for SHA-1 calculations. The figure below shows one example of differences in the code. Although the NEON code is longer and more complex, the ability to run four calculations concurrently outweighs increases length.

C

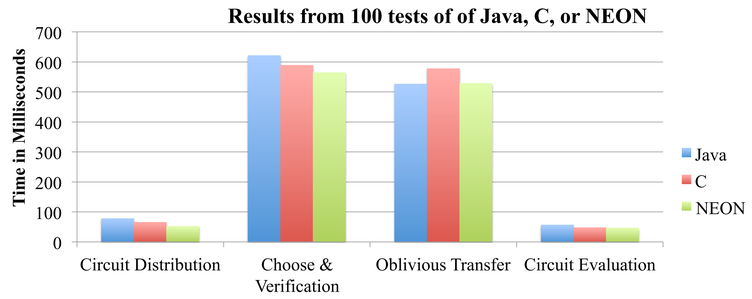
```
ldr     r2, [sp, #8]
ldr     r3, [sp, #52]
eor.w   r2, r2, r3
ldr     r3, [sp, #28]
eor.w   r2, r2, r3
ldr     r3, [sp, #20]
eor.w   r3, r2, r3
str     r3, [sp, #92]
ldr     r3, [sp, #92]
mov.w   r3, r3, ror #31
str     r3, [sp, #20]
```

NEON

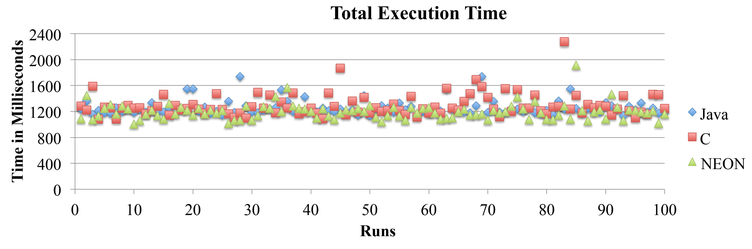
```
vld1.32     {d6-d7}, [r1]
vld1.32     {d2-d3}, [r4]
add         r4, sp, #232    ; 0xe8
vld1.32     {d4-d5}, [r4]
veor        q1, q1, q2
vld1.32     {d4-d5}, [r2]
veor        q2, q2, q3
veor        q1, q1, q2
vshl.s32    q2, q1, #2
vand        q1, q1, q0
vshr.u32    q1, q1, #30
vorr        q1, q2, q1
vmov.32     r1, d2[0]
str         r1, [sp, #296]
vmov.32     r1, d2[1]
str         r1, [sp, #300]
vmov.32     r1, d3[0]
str         r1, [sp, #304]
vmov.32     r1, d3[1]
str         r1, [sp, #308]
```

Results

For our evaluation we split the program times into four sections: 1. Circuit distribution, 2. Circuit choosing and verifying, 3. Oblivious transfer, and 4. Circuit evaluation.



Both the verifying and evaluation parts of the program where SHA-1 is used show improvement. The time required for circuit verification decreased by 5.2% for the C version and 9.2% for the NEON version. The circuit evaluation time decreased by 14.5% for the C version and 17.8% for the NEON version. No improvement was expected in the other areas of the program. The times were highly variable in the total due to network latency. We are exploring solutions to reduce this variability.



Future Work

We are aiming to continue optimizing the app by targeting the file reading section next. We believe that rewriting the input functions as native code will increase performance.

One the main difficulties with taking a normal program to the current mobile environment is the lack of dedicated memory to given the program. We hope to optimize the compiler to be memory friendly and work with complex programs on the phone.