Privacy Preserving Computations on Smartphones

Benjamin Mood and Kevin Butler

OSIRIS Lab, Computer & Information Science Department, University of Oregon

Why Privacy Preserving Computations on Phones?

Imagine that you meet someone at a party. You want to find out if you know anyone in common but don't want to reveal your whole list of friends and associates. With privacy-preserving computing, both of you can use your smartphones to automatically determine based on your address books whether you share any common contacts, *without either of you revealing your contact list to the other!*

Through computationally-intensive cryptographic protocols, privacy-preserving computing allows the ability to send information from devices as a query or as input to a distributed computation, without revealing the details of those inputs. This can allow for using services such as cloud computing, which take advantage of the computing power and storage available in the cloud, without the cloud having to know what data you are computing over.

Types of Privacy-Preserving Computations

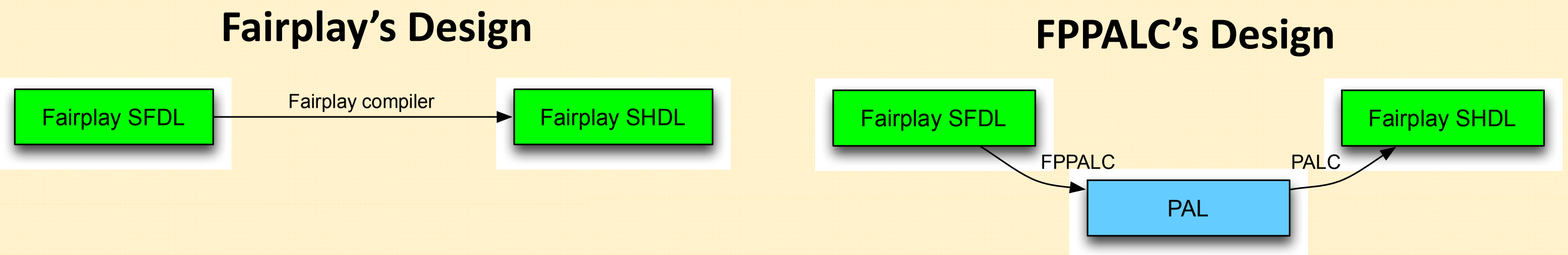
A wide range of applications can make use of privacy-preserving computing techniques. These include applications such as scheduling meetings without revealing personal calendars and discovering nearby friends without revealing current location. Multi-party applications are also possible, such as private voting and auctions where individual bids are not publicly revealed. To date, though, these computations have been too inefficient to be run in the resource-constrained smartphone environment.

Secure Function Evaluation

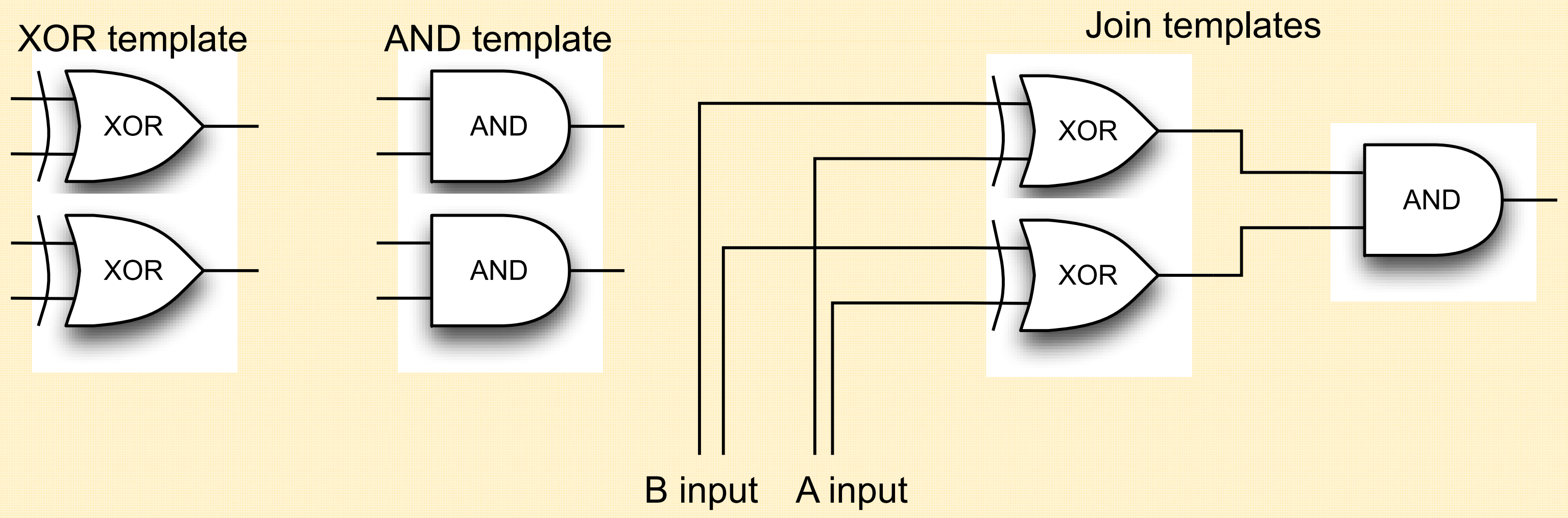
We have been exploring a type of privacy-preserving computation called *secure function evaluation*, which can use garbled circuits. These are virtual logic circuits that have encrypted values for output values in the truth tables. Only the proper input values can decrypt the output.

The standard way to create and evaluate these circuits is with software called *Fairplay*. The amount of memory needed to create the circuits, though, can run into hundreds of megabytes, more than what is available on a smartphone.

Through efficient *circuit templating* techniques where we assemble precompiled circuit functions based on a general language we designed (PAL), we can generate these circuits directly on the phone itself. Our on-phone compiler, called FFPALC, takes in files written in Fairplay's language and generates more efficient circuits.

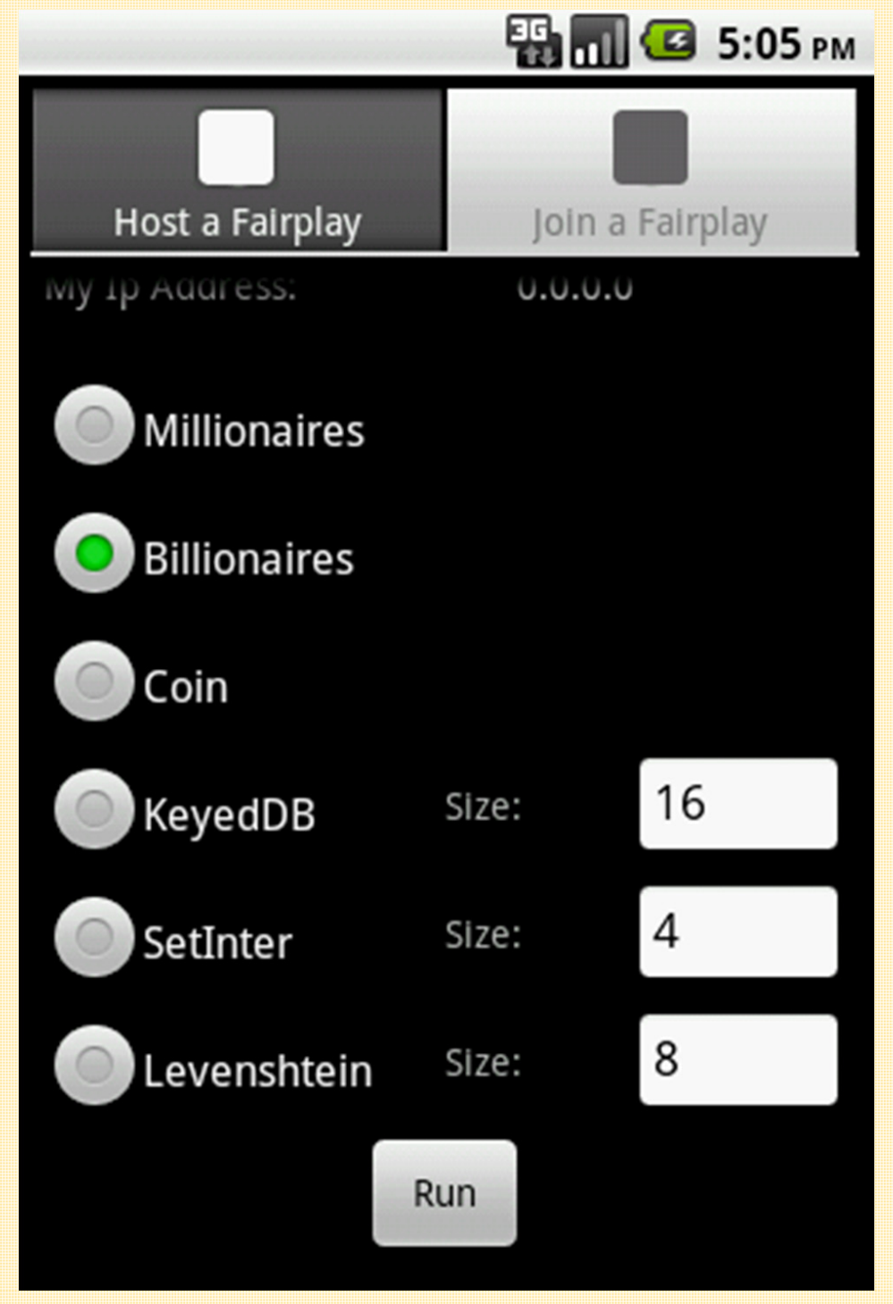


Circuit templating involves the creation of basic circuits which can then be joined to create a large program, much the same way a programmer uses a variety of statements to make a single program. These templates can then be linked together to form a larger circuit. For instance, the XOR circuit template is a template which sets up a bit by bit XOR for a variable. These templates scale up or down according to the desired bit length.

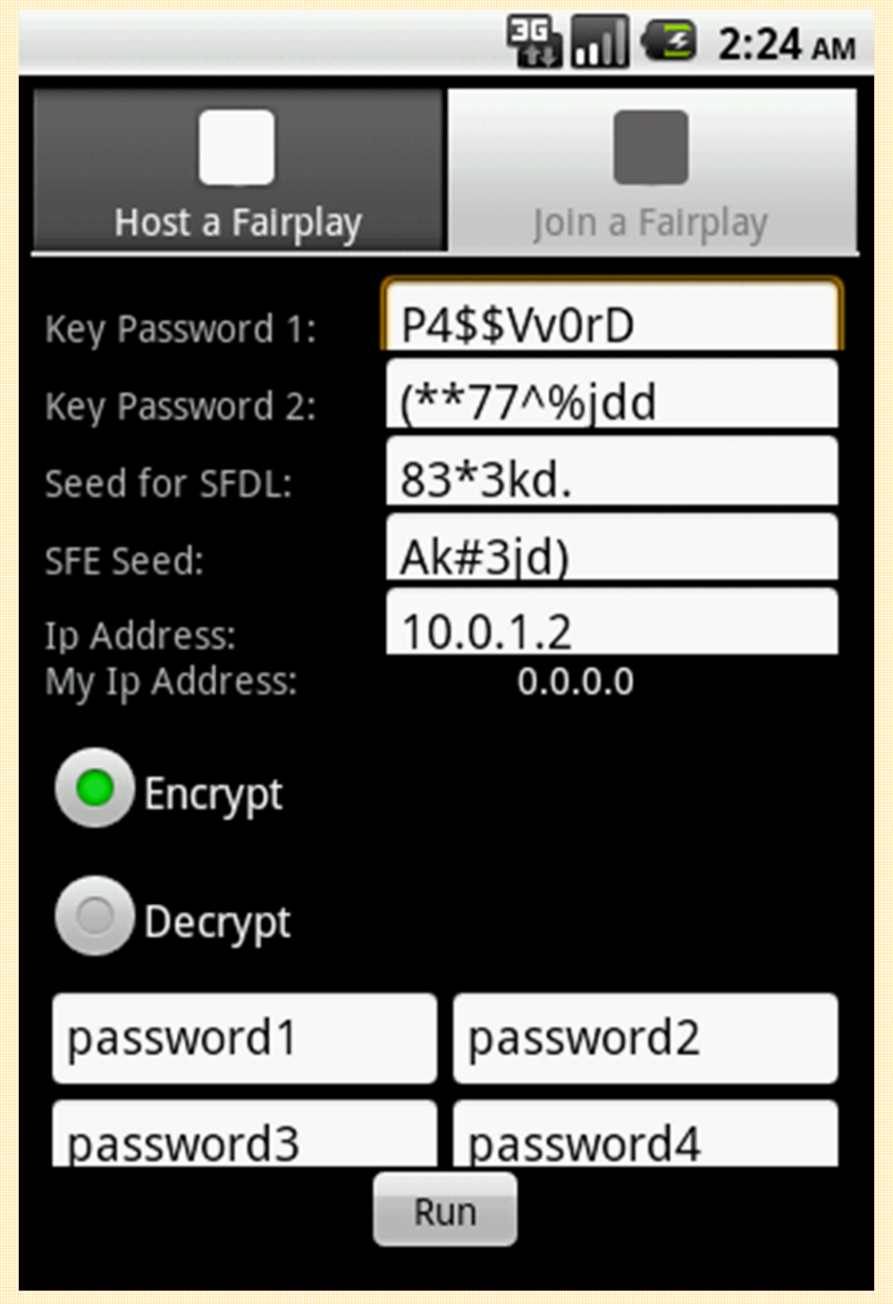


Our contribution is the creation of the fast compiler FFPALC. This compiler takes in a high level language (SFDL) and transitions to the circuit level language (SHDL) without the need for creation of a large unoptimized circuit in-between between the two languages.

We compared FFPALC and Fairplay's performance on production HTC Thunderbolt smartphones, and found memory usage was reduced by up to 95%. We designed multiple applications to use our compiler. Pictured below are an editor application (left) and a decentralized "password wallet" app for Android (right).



○ Millionaires
● Billionaires
○ Coin
○ KeyedDB Size: 16
○ SetInter Size: 4
○ Levenshtein Size: 8



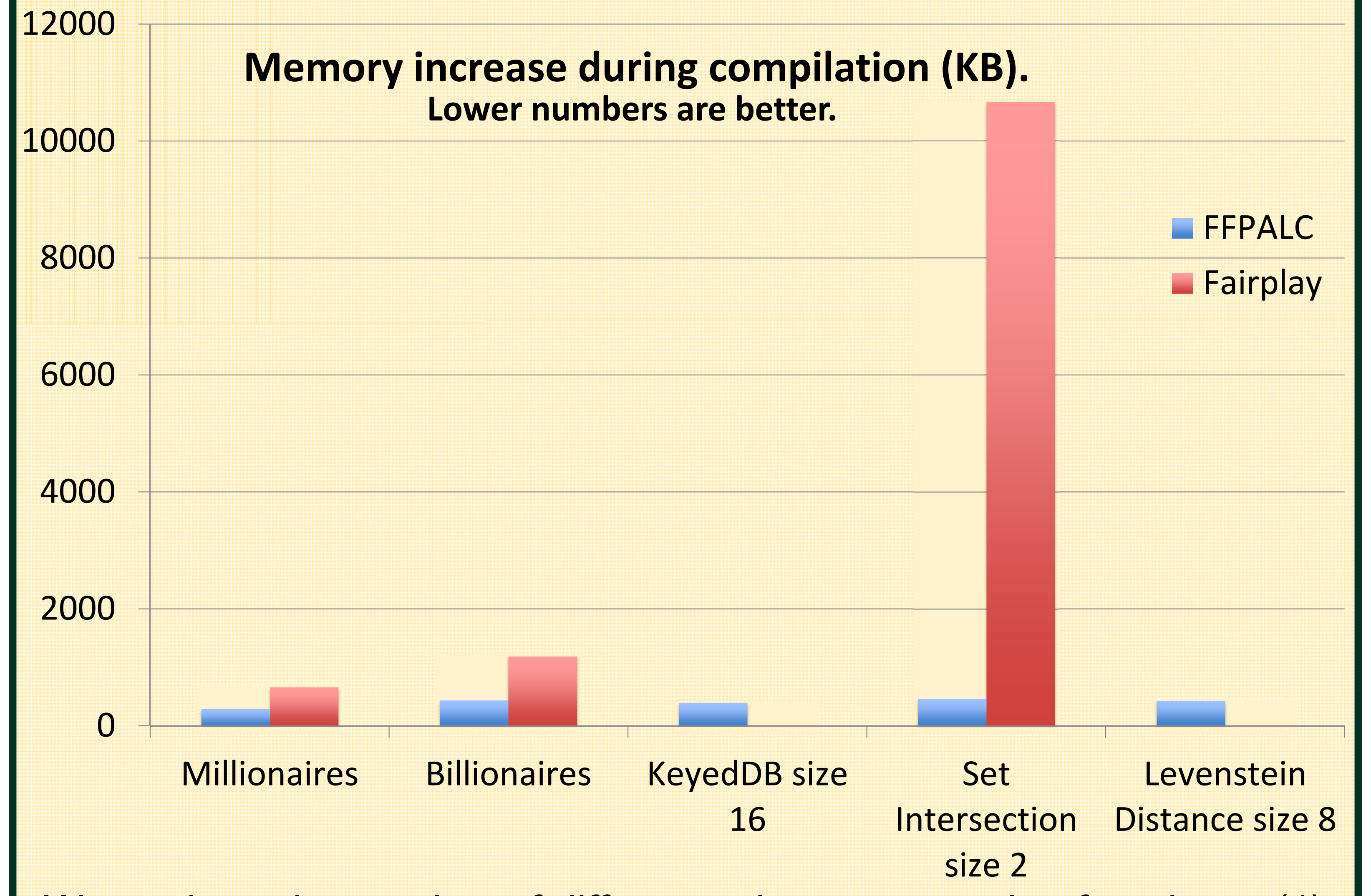
Key Password 1: P4$$Vv0rD
Key Password 2: (**77^%jdd
Seed for SFDL: 83*3kd.
SFE Seed: Ak#3jd)
Ip Address: 10.0.1.2
My Ip Address: 0.0.0.0

● Encrypt
○ Decrypt

password1 password2
password3 password4

Evaluation

We evaluated multiple programs between the two Android phones. Our compiler was faster and used far less memory. Our FPPALC compiler uses vastly less memory and allows for faster execution.



We evaluated a number of different privacy-preserving functions: *(1) Millionaires* compares two 4-bit numbers between two parties to see which is longer; (2) *Billionaires* compares a much larger 32-bit number; *(3) Keyed Database* accesses a database through a query hidden to the database owner; *(4) Set Intersection* finds common elements between two sets without revealing either set; *(5) Levenstein Distance* determines the degree of difference between two hidden words. Note that problems 3 and 5 could not even run without our compiler because the memory usage was large enough to overflow the heap on the smartphone.

Conclusion

These results demonstrate that through a carefully designed compiler, compilation of a general language to Boolean circuits can happen in a resource limited environment. In other words, privacy preserving computation is now far more feasible given our new techniques and compilers, which allow for generalized computations in resource-constrained environments. We continue to explore how to make these computations even less memory-intensive and faster to execute, so that we can fully deploy full privacy-preserving applications and infrastructures in the mobile computing environment.