Privacy-Preserving Applications on Smartphones

Yan Huang Peter Chapman David Evans
University of Virginia
http://www.MightBeEvil.org/mobile/

Abstract

Smartphones are becoming some of our most trusted computing devices. People use them to store highly sensitive information including email, passwords, financial accounts, and medical records. These properties make smartphones an essential platform for privacy-preserving applications. To date, this area remains largely unexplored mainly because privacy-preserving computation protocols were thought to be too heavyweight for practical applications, even for standard desktops. We propose using smartphones to perform secure multi-party computation. The limitations of smartphones provide a number of challenges for building such applications. In this paper, we introduce the issues that make smartphones a unique platform for secure computation, identify some interesting potential applications, and describe our initial experiences creating privacy-preserving applications on Android devices.

1 Introduction

Secure computation enables two or more mutually distrusting parties to collaboratively evaluate functions without revealing their inputs to the other parties. It eliminates the need for the distrusting parties to agree on a fully trusted third party who can receive both party's data and perform the computation, instead allowing the participants to collaboratively evaluate a function without revealing their inputs to the others. In the 1980s, Yao proposed a generic solution to secure function evaluation using *garbled circuits* [22]. The basic idea is to implement arbitrary truth tables by first representing every bit with a κ-bit key (where κ is a security parameter), then "simulating" the evaluation with encrypt and decrypt operations on those keys. Lindell and Pinkas [16] provide a detailed description and security proof of the technique. The secure computation starts by using an *oblivious transfer* (OT) protocol [13, 18] to allow the circuit generator to send the initial keys representing the evaluator's private inputs to the evaluator without learning their private inputs.

The research community has historically viewed garbled circuits as too inefficient for practical applications. However, recent work (including the *free-XOR* technique [15], the *Garbled Row Reduction* technique [20], and the scalable and efficient framework upon which this work builds [9, 12]) has brought significant efficiency improvements in generating and evaluating garbled circuits over previous implementations [12]. The main impediment to practical applications of the garbled circuit technique in previous works (most of which were built on Fairplay [17]) was the need to generate and keep in memory the entire circuit, which becomes huge for any interesting problem. Our framework overcomes this by pipelining circuit generation and evaluation. Secure computation has reached the point where practical and useful privacy-preserving applications can now be envisioned and implemented even using smartphones.

Smartphones provide unique opportunities and challenges for secure computation. They are personal devices, containing the most sensitive private information including phone calls, emails, contacts, documents, and financial and medical records. Moreover, mobile devices are increasingly used in two-factor authentication schemes and even payment systems. We can leverage this to create privacy-preserving applications not feasible on traditional platforms. On the other hand, the processing power and memory available on mobile devices is still orders of magnitude less than what is available on typical laptops, and the computation that can be done on battery-powered mobile devices is severely limited by the energy available.

Threat Models. In secure computation, a *semi-honest* adversary is assumed to always follow the protocol specification but attempts to learn additional information from observing the protocol execution. In contrast, a *malicious* adversary can deviate arbitrarily from the protocol

specification in arbitrary ways such as providing faulty garbled truth table entries for some execution paths. In many scenarios, it suffices to assume *covert* adversaries [8], who can deviate from the protocol to cheat but will be caught by a good probability (say $\frac{1}{2}$ instead of negligibly close to 1).

Contributions. In this work, we explore the design space of secure computing applications on smartphones and suggest various approaches to deal with both the efficiency and security issues by leveraging the unique properties of the mobile computing platform. Section 2 highlights the potential for secure computing applications on smartphones and summarizes our initial efforts developing secure two-party computing applications on Android devices. We find that the limited processing power instead of the bandwidth poses the biggest performance hurdle. Section 3 discusses the technical and social challenges that must be addressed to make secure computation successful on the mobile platform.

2 Applications

Secure computation can enable a wide range of privacy-preserving mobile applications. Smartphones stay with their owners all day and are filled with private information. Secure computation techniques serve as the preferable way to use the private data with minimal compromise of users' privacy. On the other hand, secure computation usually requires all participants to be online to run the secure protocols, whereas naïvely keeping the agent programs always online can lead to excessive leakage of users private data due to repeated protocol execution. With smartphones, the dilemma is alleviated since users can either conveniently activate a protocol agent explicitly, or can employ a strategic automated procedure to decide whether to join a secure computing protocol.

Two-Party Applications. There are many interesting and useful privacy-preserving applications that are especially well-suited to smartphones because of their ubiquity, mobility, and access to personal data. For example, users could compute a joint function over their frequent contacts, Facebook and LinkedIn acquaintances, or the locations they have been. In another application, business associates could easily use their phones to allocate a common slot on their timetables for a collaborative meeting without leaking other information about their private schedules. Moreover, with computations on location data, people could be able to quickly discover nearby friends without otherwise sacrificing location privacy. Retailers could implement targeted advertising where merchants keep their special offers secret while customers would not need to sacrifice their private purchasing history or personal preferences.

Multi-Party Applications. Many privacy-preserving applications such as private event scheduling are even more useful in multi-party scenarios. Many classic multi-party secure computing applications such as privacy-preserving voting, auctions, peer-ranking and stable matching [7] would also be useful on smartphones. While our prototype applications target two-party scenarios, garbled circuit protocols can be extended to support multiple parties [6]. Thus, our approach could be extended to enable privacy-preserving computations involving several participants. Many additional technical and engineering challenges would need to be addressed to support large multi-party computations. For example, to pair the devices, we might want to use location proximity metrics (based on GPS, or a communication channel like Bluetooth) to conveniently setup a multi-party computation for a set of nearby devices.

For the rest of this paper, we focus on two-party applications. Next, we describe our experiences in developing two prototype secure computations applications where participants explicitly invoke and interact with the applications. The following subsection speculates on opportunities for latent secure computations that execute automatically in the background.

2.1 Explicit Secure Computations

We have built two prototype privacy-preserving applications for Android devices, both of which involve explicit interactions from both participants. The prototypes use the garbled circuit framework from Huang et al. [12] in a straightforward way, and use Wi-Fi for all communication. The Java framework and library allow a developer to translate the secure components of the target application into Boolean circuits from which the library facilitates the secure computation. They demonstrate the feasibility of secure computation on smartphones, but are only a first step towards exploring the full potential of secure computation on smartphones.

Preliminary results show that the mobile devices are roughly 1000 times slower than typical desktops. Running on Android Nexus One phones, the protocols execute at a speed of approximately 100 non-free gates per second compared to 100,000 gates per second for desktops [12]. Even this rate, however, is fast enough to enable some interesting secure applications. In Section 3 we discuss methods that will enable much faster secure computations on current and future smartphones.

Private Set Intersection. Private set intersection enables many interesting privacy-preserving applications. For example, two people who meet at a conference could securely discover mutual contacts without reveal-

ing their address books, or two new acquaintances can find their common friends, hobbies, or recently visited places without exposing other information about their unshared friends or details about their movement. To investigate the feasibility of secure computation on mobile devices, we developed an Android application to discover the common contacts of two users while preserving the privacy of unshared contacts.

Previous approaches to private set intersection either use homomorphic encryption to evaluate a polynomial that encodes the set elements [5] or employ a secure encryption protocol. Huang et al. developed a series of methods to compute set intersection more efficiently using garbled circuits [11]. For example, with the Bit-AND scheme each set element is mapped to a bit in the bit-vector, the set intersection is computed by simply using garbled AND gates. Despite its simplicity, this is suitable and efficient for many real world applications where the element space is relatively small (e.g. $\leq 2^{16}$). Using Bit-AND, computing the common elements from a set of about 100 candidates can be finished almost instantly.

For the large element space of email addresses and phone numbers, the Bit-AND-based scheme is impractical, and so we implemented a more complex private set intersection scheme, called Sort-Compare-Shuffle (SCS) [11] to perform the computation in $\Theta(n \log n)$ time, where n is the number of contacts to be compared.

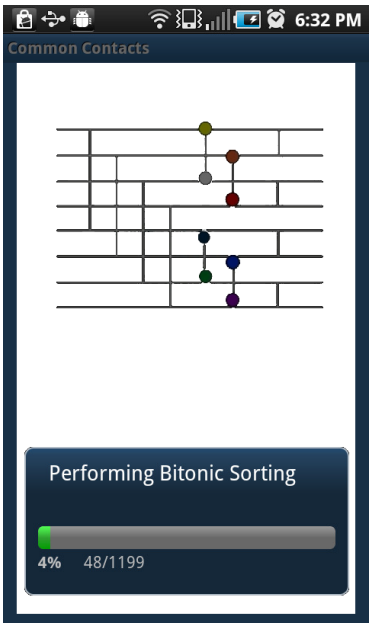

Figure 1: Screenshot of *CommonContacts* Running Intersection Computation. The background video illustrates the computation phases such as the bitonic sorting operation shown in the image, but reveals nothing about the specific execution.

The *CommonContacts* application [10] can compute the contact set intersection as long as participating devices are running the software and connected to the same Wi-Fi network. The unique identifier of each contact (the email address or phone number) is hashed to a 24-bit value, and the result is the intersection of the sets of hash values. Each client will display the result by mapping the hash values in the result back to the corresponding contact's identifier.

Figure 1 shows a screenshot of *CommonContacts* performing the comparison. Table 1 shows the time cost. Execution takes on the order of minutes, depending on the speed of the devices and set sizes. Section 3 discusses some of the reasons the performance on smartphones is so much worse than on current desktops, and opportunities for improving it.

Similar to our common contacts application, De Cristofaro et al. [3] recently studied the *Private Contact Discovery* problem which focuses on enabling peer-to-peer common certified contact discovery. Based upon the RSA assumption on *safe moduli* in Random Oracle Model, they develop a custom Contact Discovery Scheme (CDS) that includes protocols to add, revoke, and discover contacts. In contrast to our approach using generic garbled circuits, customized cryptographic protocols require new security proofs and can be difficult to adapt to other applications. On the other hand, custom protocols could provide better performance for targeted applications. De Cristofaro et al. report taking 5 seconds to run the "discover" protocol with 100 contacts on an ARMv7 processor. However, it is hard to compare the results directly since their scheme incurs periodic extra costs to execute the "add" and"revoke" protocols. Additionally, it is also unclear how expensive it is to retrieve the public keys of all contacts involved in a discovery.

Personal Genetics. The cost of genome sequencing is dropping to the point where it may soon be common for individuals to store their own genome on their smartphones. This could enable secure computing applications that allow friends or even strangers to search for kinship relationships (e.g., discovering you are likely to be third cousins) or even *genetic dating* applications that would allow two potential partners to estimate the risk that their off-spring would have certain hereditary diseases without otherwise exposing their genomes. Affordable services already exist to perform a computationally

Set size	32	64	128	256
Time (seconds)	68	134	285	598

Table 1: Total Execution Time for *CommonContacts* for Different Input Sizes (running on Nexus One phones).

insecure genomic analysis [14, 19] by signing legal contracts. Many different methods exist to estimate the risks of hereditary diseases, varying in the level of complexity. In a simple model, genetic diseases are determined by recessive alleles, meaning that if both parents are carriers their children will suffer from that disease. This functionality can obviously be computed by AND-ing both parent's carrier bit. Using a benchmark prototype implementation on our Nexus One phones, it takes about 7 seconds to estimate the risk of 25 genetic diseases.

2.2 Background Secure Computations

Our demo applications target scenarios in which both parties consent to the computation by deliberately running the process. These scenarios become more practical with mobile devices because users can spontaneously run the computations on-demand. However, applications can further leverage the ubiquitous and always-available nature of mobile devices to perform computations periodically or constantly on behalf of the owner.

For example, the genetic kinship application can constantly run in the background and notify users when a (possibly unknown) genetic relative is nearby. In a similar scenario, a social application can ping nearby people searching for shared interests or hobbies. Real-time marketing offers utility to both consumers and retail, but privacy concerns can limit recommendations and personalization. A privacy-preserving targeted advertising system would leverage browsing and shopping history to allow physical retail locations to make specialized discounts directly to a customer's device upon entering the store. Consumers may not want to share their full interests and shopping history with the merchant, but may be willing to accept personalized special offers based on a secure computation using this information.

One issue with latent secure computations, is that each protocol execution may leak some information about the user's private data. Hence, we need to cap the number of protocol executions to ensure the overall leakage of the private information is acceptable. Depending on the nature of application, a sophisticated strategy could be employed to decide whether to participate in a protocol execution based on a number of metrics such as the peer's ID, time of last execution, or frequency and results of previous executions. Since we are targeting smartphones, it is also possible to enforce execution requirements based on the current location.

An even more robust and interesting discretion rule could use the wireless carrier, a jointly trusted entity, to ensure only executing the protocol with "legitimate" peers (e.g., peers in my phone book or peers I talked to within last 3 months). Another approach is to use application-specific self-auditing schemes to maintain low information leakage. Privacy-preserving applications could include relatively simple logic (implemented by garbled circuits) into the computation to roughly estimate how much information about the user's private input can be leaked by the imminent output. As an example, for private set intersection it is advisable to add logic to the circuit that checks the size of the result. If that size exceeds some prescribed threshold, the protocol terminates without revealing the result since this result would leak too much information about the owner's data (and may indicate that the peer is probing by misrepresenting its own set).

3 Challenges

The main challenges in implementing privacy-preserving applications on mobile devices stem from the limited resources (including computation power, bandwidth, and energy) available on these devices.

Performance. Our early prototypes indicate that the main performance bottleneck on mobile devices is the limited processing power rather than the network bandwidth, which averages to about 3 KB/s (far less than the capacity of any popular Wi-Fi network).

Our prototype implementation uses the MD5 hash function for the encryption scheme. Due to the lack of good hardware support for fast cryptographic operations on today's Android phones, the processing speed is about 1000 times slower than on current desktops. This performance could be substantially improved either by accessing a cryptographic hardware module, or by finding a cipher algorithm that is more efficient for mobile devices.

In addition, we also observe substantial footprint of the JVM's garbage collection thread. Our code uses many BigInteger objects, which is an *immutable* datatype provided by the Java API. Combined with the fact that smartphones have very limited memory, this results in very frequent halts to reclaim storage used by old BigInteger objects. We anticipate improvements to our implementation that will avoid this problem by using a mutable datatype and explicitly managing the memory needed to represent the garbled table entries.

Circuit execution is inherently easy to parallelize so that multiple cores, especially those on the GPU, can easily bring about a large speedup. Android 3.0 supports multiple cores and several devices are now on the market with dual-core processors. It may also be helpful to leverage the Renderscript API available in Android 3.0, which allows low-level, high performance execution on the device CPU or GPU to increase both concurrent operations and per instruction work [1].

Pairing. The way to establish network connections can be an important design decision affecting application

performance during the computation and user experience in device pairing. As a fallback strategy, devices could be paired through a third-party and communicate over the Internet. For applications that are better run with peers physically nearby, the best choice would be communicating directly through Wi-Fi, though few devices on the market support ad-hoc Wi-Fi networks and thus would require an an existing access point. Bluetooth communication has sufficient bandwidth and is another alternative for peer-to-peer communication mechanism, but pairing can be cumbersome and requires direct user action on Android. To supplement other technologies and smoother pairing of devices, users could utilize the *Near Field Communication* (NFC) sensors included in newer smartphones. Another strategy is to take advantage of trusted relationships with carriers to either set up Wi-Fi connections with known peers, or to make connections directly using the cellular network.

Developing Applications. Most developers are unaware of secure computation. New industry education efforts and the development of a wide-variety of tools can support adoption of privacy-preserving applications for smartphones.

The secure computation design process we envision begins with the development team identifying secure and insecure aspects of the application with manual inspection or the help of tools, possibly using taint-tracking techniques to identify the security requirements of applications. Then to implement the secure functions, developers can use specific primitives found in the literature or a general framework like the one we have developed.

Our Java-based generic secure-computation framework requires developers to translate their desired function into a binary circuit from which we can execute. We provide a series of basic gates and sub-circuits to serve (AND, OR, XOR, adders, comparators, etc.) as building blocks with more complex components being developed and shared over time. This circuit-level approach enables important optimizations that can dramatically reduce the number of garbled gates that must be generated and evaluated [12], but a higher-level representation (such as the Algol-like language used by Fairplay [17]) may be more accessible to typical developers if compilers can be developed that produce sufficiently efficient circuits from high-level representations. Eventually, we hope it will be possible to develop tools that automatically produce efficient secure computations from Java programs annotated with information about what data is private.

Lastly, it is important to be aware of what can be inferred about the private data from the output of a secure computation. For example, a naïve set intersection implementation could allow one party to use the universal set as the input, revealing the other participant's entire set. It may be possible for development tools to eventually include techniques for automatically generating auditing circuits that check the amount of information that would be leaked before a result is revealed.

Stronger Adversaries. Our prototype applications assume semi-honest adversaries, a very weak threat model. However, efficient protocols against stronger adversaries remain to be developed. In this sense, known techniques (either traditional *cut-and-choose* or *commit-and-proof*) to thwart malicious adversaries are too expensive to be feasible on smartphones in the foreseeable future. As an alternative, we could consider applying *software-based attestation* techniques [21] to ensure each party runs an expected implementation of the protocol. Because smartphones have more fixed hardware and software than traditional desktops, software-based attestation may be more feasible for smartphones than more complex platforms. In addition, it could also be a promising direction to develop protocols based on *commodity-based cryptography* [2, 4] where a third party (such as the wireless carrier) is trusted to providing private-data-independent random strings that satisfy certain property but does not receive any private data.

Users. Secure function evaluation is a difficult and somewhat paradoxical concept for typical smartphone users. Wide acceptance depends on both careful user interface design, integration with smartphone platforms, and significant user education efforts to convince users how a secure computation is different from providing their data to a peer or third party. Even with good interfaces, users will need to understand what it means to install and execute a privacy-preserving application. Additionally, running applications should offer clear (graphical) indications on what data is protected during the computation and differentiate themselves from other typical programs. Perhaps standard permissions could be extended with new permissions that allow an application to access particular private data, but to only use it in network protocols that use secure computation to prevent leaking. We also envision the emergence of a privacy-focused application markets where a trusted party vouches for privacy properties of applications built using a standard framework.

4 Conclusion

Smartphones are becoming an essential computing platform, and the devices that store many individual's most private data. Combined with the significant improvements in the efficiency of garbled circuit execution, this presents an exciting opportunity for a host of new privacy-preserving applications that leverage the mobility, location-awareness, data accessibility, and the relationships people have with their smartphones.

Availability

The secure computation framework is available at http://mightbeevil.org/framework/ under the MIT license. Our sample Android applications are available at http://mightbeevil.org/mobile/.

Acknowledgments

This work was partially funded by the National Science Foundation and a MURI award from the Air Force Office of Scientific Research. We thank Google for providing the Nexus One smartphones. The authors thank Sang Koo for helping with the *CommonContacts* application user interface. We also thank Jonathan Katz for useful advice on this project.

References

[1] Android Developers. Renderscript. http://developer.android.com/reference/android/renderscript/package-summary.html.

[2] Donald Beaver. Commodity-based Cryptography. In *ACM Symposium on Theory of Computing*, 1997.

[3] Emiliano De Cristofaro, Mark Manulis, and Bertram Poettering. Private Discovery of Common Social Contacts. In *Applied Cryptography and Network Security*, 2011.

[4] Wenliang Du and Zhijun Zhan. A Practical Approach to Solve Secure Multi-party Computation Problems. In *New Security Paradigms Workshop*, 2002.

[5] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, 2004.

[6] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play Any Mental Game. In *ACM Symposium on Theory of Computing*, 1987.

[7] Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, USA, 1989.

[8] Carmit Hazay and Yehuda Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *Theory of Cryptography Conference*, 2008.

[9] Yan Huang. Fast Secure Computation Framework. http://www.MightBeEvil.org/framework/, 2011.

[10] Yan Huang, Peter Chapman, Sang Koo, and David Evans. *Common Contacts* Android App. http://mightbeevil.org/contacts/, 2011.

[11] Yan Huang, David Evans, and Jonathan Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? http://www.cs.virginia.edu/yhuang/pubs/psi.pdf, 2011.

[12] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security Symposium*, 2011.

[13] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO*, 2003.

[14] KnowYourGenes.org. Genetic Testing Information from the Genetic Disease Foundation. http://www.knowyourgenes.org/.

[15] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *International Colloquium on Automata, Languages and Programming*, 2008.

[16] Yehuda Lindell and Benny Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology*, 2009.

[17] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—A Secure Two-party Computation System. In *USENIX Security Symposium*, 2004.

[18] Moni Naor and Benny Pinkas. Efficient Oblivious Transfer Protocols. In *ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[19] National Center for Biotechnology Information. GeneTests. http://www.ncbi.nlm.nih.gov/sites/GeneTests/.

[20] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure Two-Party Computation is Practical. In *ASIACRYPT*, 2009.

[21] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep K. Khosla. SWATT: SoftWare-based ATTestation for Embedded Devices. In *IEEE Symposium on Security and Privacy*, 2004.

[22] Andrew C. Yao. How to Generate and Exchange Secrets. In *Symposium on Foundations of Computer Science*, 1986.