A simple protocol for secure circuit evaluation

(Extended abstract)

Martín Abadi
DEC Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301

Joan Feigenbaum
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract: We present a simple protocol for *secure circuit evaluation*. In its most basic formulation, the protocol enables two players C and D to cooperate in the computation of $f(x)$ while D conceals her data x from C and C conceals his circuit for f from D.

1. Introduction

This paper describes a protocol for *secure circuit evaluation*. We think of player C as a computer that runs a secret program for f and player D as another computer with some confidential data x. Suppose that D wishes to compute $f(x)$ but does not have an algorithm for f, and that C is willing to let D use his algorithm but does not want to reveal it. Alternatively, suppose that C has an algorithm, and possibly some confidential data as well, but needs D's confidential data in order to perform the computation. In its most basic formulation, the protocol enables C and D to cooperate in the computation of $f(x)$ while D conceals her data x from C and C conceals his circuit for f from D; in fact, the circuit is hidden from D in an information-theoretic sense.

For example, suppose that C is a computer and that D is one of C's users. Typically, when D wants to log in, she submits a name and a password, and C runs a program that checks whether they are correct. Of course, C does not have to divulge his program. Our protocol provides a way for D to log in without revealing her name or her password.

This protocol is very simple and easy to analyze, and it has many of the desirable properties of more complex ones. We demonstrate that only simple tools are needed to conceal circuits and data. The simplicity of our scheme makes it easy to focus on specific new themes, such as the tradeoff between *hiding* in an information-theoretic sense and *encryption* under complexity-theoretic assumptions.

We discuss the security properties of the protocol, and, in general, discuss hiding vs. encryption. For instance, we show that it is not possible to hide everything about the computation except the value $f(x)$; a similar result is suggested in [CDvdG]. We also study the efficiency of the protocol. We show that the number of rounds of communication that it requires is less than or equal to the depth of C's circuit.

The basic protocol can be extended in a number of useful ways. C can hide some secret

data as well as his circuit. The players can compute a sample point of a distribution, instead of a functional value, just by including some random inputs. C can cooperate with several other players, $D_1, \ldots, D_m$, each with some private data. We discuss variations on the basic protocol. The issue of cheating is different in frameworks in which some secrets are hidden rather than encrypted. We discuss what forms of cheating might arise and how they can be thwarted.

In the next section we describe the protocol. In Sections 3 and 4 we consider security and efficiency issues, respectively. In Section 5, we sketch two variants of the protocol for many players. We study cheating in Section 6. In Section 7, we describe some recent related work. We conclude with an open problem in Section 8.

2. A basic protocol

As a preliminary, we review one of the results from our joint work with Kilian ([AFK]).

The quadratic residuosity function takes two arguments, an integer k of the form $p \cdot q$, where $p \equiv q \equiv 3 \bmod 4$, and an integer u in the set $Z_k^*[+1]$ — the integers relatively prime to k with Jacobi symbol 1. (One can define the quadratic residuosity of a broader class of pairs of integers, but the increased generality is not needed here.) The value $r(u,\ k)$ is 1 if there is an integer a in $Z_k^*[+1]$ such that $a^2 \equiv u \bmod k$, and it is 0 otherwise.

In [AFK], we discussed the following scheme for computing the quadratic residuosity function with secret data. To conceal the instance $\langle u,\ k\rangle$, the player chooses b from $Z_k^*[+1]$ and c from $\{0,1\}$, both according to uniform distributions, and computes $v = u \cdot b^2 \cdot (-1)^c \bmod k$. Informally, we often denote v by $E(u)$. The new instance is $\langle v,\ k\rangle$, and the key is the pair $\langle b,\ c\rangle$. Let $e = r(u,k)$ and $e' = r(v,k)$. Then, to decode, that is, to compute e from e', we let e equal e' if $c = 0$, and we let e equal $1 - e'$ if $c = 1$. We denote the decoded answer by $F(e')$. Correctness follows from the facts that $u \cdot v$ is a residue mod k if and only if both u and v are residues or neither is a residue and that -1 is a quadratic nonresidue if k is of the specified form. Only k needs to be known for all the operations to be feasible in polynomial time. We gave a simple proof in [AFK] that this scheme for encoding instances of the quadratic residuosity problem allows the encoder to obtain $r(u,k)$ without revealing anything about u, in an information-theoretic sense — that is, while hiding u.

This technique for consulting an oracle with no information transfer is a building block in our protocol for secure circuit evaluation. We also use a standard technique for encrypting single bits: given a bit b, it is easy to generate a y such that $r(y,k) = b$ ([GM]). Informally, we often denote y by $Y(b)$.

In our current setting, player D has the input x, which can be written $x_1 \ldots x_n$ in binary. Player C has a circuit to compute f on inputs of length n. In the initial phase of the protocol, C sends to D the number a of AND gates in his circuit. D then generates $k = p \cdot q$, where p and q are primes congruent to 3 mod 4 and large enough with respect to a, in a sense that we make precise in the next section. Next, D encrypts her input bits $x_1, \ldots, x_n$, using modulus k as explained above. Finally, D sends k and $Y(x_1), \ldots, Y(x_n)$ to C.

We assume without loss of generality that player C's circuit consists of unary NOT gates

and binary AND gates and that it has no consecutive NOT gates. In the protocol, C simulates the evaluation of the circuit on D's input. Instead of bits b_i, C will have integers of the form $Y(b_i)$ that represent bits. We must now show how to simulate the logical operations NOT and AND using this scheme for representing bits.

When C must compute the NOT of a bit b, represented by its encryption $Y(b)$, he computes $Y(\bar{b})$. Because $p \equiv q \equiv 3 \bmod 4$, -1 is a quadratic nonresidue mod k, and thus C does not need to know b: $Y(\bar{b}) \equiv (-1) \cdot Y(b) \bmod k$.

To compute the AND of two bits b and b', represented by their encryptions $Y(b)$ and $Y(b')$, C requires the cooperation of D. C transforms b and b' further, to obtain $E(Y(b))$ and $E(Y(b'))$, which he sends to D. Then D computes $b_e = r(E(Y(b)), k)$, $b'_e = r(E(Y(b')), k)$, which she can do efficiently, because she knows p and q. Let b_3, b_2, b_1, and b_0 equal $(b_e \wedge b'_e)$, $(b_e \wedge \bar{b}'_e)$, $(\bar{b}_e \wedge b'_e)$, and $(\bar{b}_e \wedge \bar{b}'_e)$, respectively. D returns $\langle Y(b_3), Y(b_2), Y(b_1), Y(b_0)\rangle$. Since C knows whether $b = b_e$ and whether $b' = b'_e$, he can easily obtain the encrypted conjunction of $Y(b)$ and $Y(b')$ from the message: it is $Y(b_3)$ if both equations hold, $Y(b_2)$ if only the first equation holds, $Y(b_1)$ if only the second equation holds, and $Y(b_0)$ if neither equation holds.

The players follow one of two versions of the final phase of the protocol, depending upon which of them is supposed to receive the answer $f(x)$. In Case 1, C retains the answer. At the end of the gate-simulation phase, C has the encrypted bit $Y(f(x))$. He encodes this bit further by computing $E(Y(f(x)))$, which he then sends to D. Player D returns the residuosity bit $r(E(Y(f(x))), k)$. C decodes the bit to obtain $f(x)$. In Case 2, in which player D receives the answer, C sends to D the encrypted bit $Y(1) \cdot Y(f(x))$, and D decrypts to obtain the residuosity bit $r(Y(1) \cdot Y(f(x)), k)$, which is, by definition, $f(x)$.

In the following description of the protocol, the notation "P1 $\rightarrow$ P2: m" means player P1 sends the message m to player P2. The notation "P: s" means that player P evaluates the statement s.

Initial Phase:

C $\rightarrow$ D: The number of AND gates in his circuit.

D: Generate k and $Y(x_1), \ldots, Y(x_n)$.

D $\rightarrow$ C: $k, Y(x_1), \ldots, Y(x_n)$.

Gate-Simulation Phase:

NOT gate with input $Y(b)$

C: $Y(\bar{b}) := (-1) \cdot Y(b) \bmod k$.

AND gate with inputs $Y(b)$, $Y(b')$

C $\rightarrow$ D: $E(Y(b)), E(Y(b'))$.

D: $b_e := r(E(Y(b)), k)$; $b'_e := r(E(Y(b')), k)$.

$\langle b_3, b_2, b_1, b_0\rangle := \langle b_e \wedge b'_e,\ b_e \wedge \bar{b}'_e,\ \bar{b}_e \wedge b'_e,\ \bar{b}_e \wedge \bar{b}'_e\rangle$.

D $\rightarrow$ C: $Y(b_3), Y(b_2), Y(b_1), Y(b_0)$.

C: If $b = b_e$ and $b' = b'_e$ then $Y(b \wedge b') := Y(b_3)$.

If $b = b_e$ and $b' \neq b'_e$ then $Y(b \wedge b') := Y(b_2)$.

If $b \neq b_e$ and $b = b'_e$ then $Y(b \wedge b') := Y(b_1)$.

If $b \neq b_e$ and $b' \neq b'_e$ then $Y(b \wedge b') := Y(b_0)$.

Final Phase:

Case 1: C keeps the answer

C $\rightarrow$ D: $E(Y(f(x)))$.

D: $b := r(E(Y(f(x))), k)$.

D $\rightarrow$ C: b.

C: $f(x) := F(b)$.

Case 2: D receives the answer

C $\rightarrow$ D: $Y(1) \cdot Y(f(x))$.

D: $f(x) := r(Y(1) \cdot Y(f(x)), k)$.

From this description of the basic protocol, it is clear that the distinction between "data" and "circuits" is unnecessary. If C has the ability to hide a circuit, then he can also hide some private data, simply by "hardwiring" it into the circuit. Conversely, in protocols in which C has the ability to hide data, he can also hide a circuit through a detour: C can run the protocol, take f to be a universal function, and use the Gödel number of the circuit he wants to hide as input.

3. Security properties of the basic protocol

Our first result is that the initial and gate-simulation phases of the protocol hide the circuit from D, regardless of how powerful she is.

Theorem 1: *During the initial and gate-simulation phases,* C *reveals no information about his circuit except the number of* AND *gates. At the end of arbitrarily many gate simulations, possibly in different runs of the protocol, each circuit with this number of* AND *gates is equally probable from D's point of view.*

In the protocol, D's data is not hidden from C, but it is encrypted. In our terminology, this means that the security of the data follows from a hypothesis about the intractability of a computational problem. Here, the hypothesis we use is the Quadratic Residuosity Assumption (QRA).

QRA: *Suppose that k is known to be of the form $p \cdot q$, where p and q are primes congruent to* 3 mod 4, *but that the factors p and q are unknown. Suppose that $u \in Z_k^*[+1]$. Then $r(u, k)$ cannot be computed in random polynomial time.*

The QRA is ubiquitous in recent cryptographic literature. The problem of finding all four square roots of u, where $k = p \cdot q$ and $r(u, k) = 1$, is equivalent to factoring k's of that form, under randomized polynomial-time reductions.

Theorem 2: *Let c be a positive constant and a be the number of* AND *gates in* C's *circuit. Assume that $|k| > a^c$. Then, under the* QRA, *with high probability,* C *learns nothing about* D's *input bits during the initial and gate-simulation phases of the protocol.*

Thus, in the second step of the initial phase, D must choose p and q large enough so that $|k| > a^c$. That is required to *prove* security of D's input bits under the QRA; in practice, D

would choose k large enough so that it could not be factored in a reasonable amount of time using the best known factoring algorithms.

We now state our results about the information communicated during the final phase.

Theorem 3: *In* Case 1 *of the final phase,* D *learns nothing, and, under the* QRA, C *learns only* $f(x)$.

Theorem 4: *In* Case 2 *of the final phase,* C *(obviously) learns nothing, and* D *learns only* $f(x)$.

In Case 2, C multiplies the encrypted output of the circuit's final gate by a random square so that D does not learn whether the final gate is a NOT or an AND. Suppose that C did not multiply by a random square but instead sent the output of the final gate, that is, $Y(f(x))$. If the final gate is a NOT, then $Y(f(x))$ is the additive inverse, mod k, of one of the elements of some quadruple that D sent during the protocol. If the final gate is an AND, then $Y(f(x))$ is itself one of the elements of the last quadruple that D sent. The probability that $Y(f(x))$ is both of these is exponentially small; thus translation by a random square is necessary in order to conceal whether the final gate is a NOT.

Why would one want a protocol that hides some secrets while others are just encrypted? In other words, if we believe the QRA, then why do we need to hide some secrets unconditionally? First, as demonstrated above, hiding can be conceptually simpler than encryption, and it is not necessarily more expensive in terms of computational resources, even though it is certainly more secure. Second, there may be a disparity in power among players. We may want to treat only certain players as though they could crack cryptosystems based on intractability assumptions. Finally, there may be a disparity in the sensitivity of different secrets. For instance, some data is so ephemeral that by the time it is decrypted it is no longer useful. On the other hand, a circuit (or a secret algorithm) may need to be used repeatedly over a long period.

Thus it is natural to ask whether we could evaluate circuits while hiding all secrets. For instance, can we transform our basic protocol so that D's data, as well as C's circuit, is hidden? It is clear from our remarks at the end of Section 2 about the equivalence of "data" and "circuits" that we can trade off hiding and encryption, i.e., that we can transform the protocol so that D's secret is hidden and C's is encrypted. Can we go further and design a protocol for secure circuit evaluation in which only one bit of information (say, the functional value $f(x)$) is exchanged? Our next result says that such protocols do not exist.

Theorem 5: *No protocol that hides all secrets and communicates only one bit of information can compute* $f(x)$ *accurately for all* f *and all* x.

To prove Theorem 5, we show that, even for some very simple functions f, one can construct runs in which a protocol that hid both C's and D's secrets would not compute $f(x)$ correctly. We give this and all other proofs in the full paper.

4. Efficiency considerations

The basic protocol of Section 2 is conceptually simple and amenable to implementation. Like Galil *et al.*'s protocol, and unlike Yao's original protocol, our scheme requires the generation of only one large modulus k. The amount of communication required is one message in each direction for every AND gate, plus one exchange of messages during the initial phase and one during the final phase. This compares favorably with the requirements of previously published schemes.

We can reduce the number of messages even further by having C send simultaneously the pairs of hidden inputs $E(Y(b))$, $E(Y(b'))$ of all the AND gates at a given level in his circuit. In order to avoid leaking some additional information about the structure of the circuit, e.g., the maximum number of AND gates at a level, C could add some extra questions to each round. Note that this reduces the number of *rounds* of communication between C and D to the nesting depth of AND gates in the circuit, but it does not reduce the total length of the messages exchanged; so it may or may not be a worthwhile optimization.

There is no need to restrict ourselves to the primitives NOT and AND; other gates, such as OR's, can be implemented completely analogously. In general, the implementation of a logical primitive entails communication between C and D (as in the case of our implementation of AND), but some useful primitives (like NOT) can be implemented with no communication. For instance, the EQUAL gate, which tests for the equality of two bits, can be simulated by C alone: $Y(b) \cdot Y(b')$ is a residue mod k if and only if $Y(b)$ and $Y(b')$ are both residues or neither is a residue, i.e., if and only if $b = b'$. Hence the bit $b = b'$ can be represented by $Y(b) \cdot Y(b') \bmod k$, which C can compute without help from D.

In summary, the basic protocol is a reasonable starting point for an implementation of a secure circuit-evaluation system. It performs particularly well on circuits with many gates that do not require communication, and with a shallow nesting of gates that require more communication.

5. Secure computations among many players

In this section we briefly discuss the extension of the basic protocol to computations among an arbitrary number of players. Our discussion is brief, because similar extensions have been thoroughly studied for related protocols. The basic protocol can be generalized to the case in which there are m databases $D_1, \ldots, D_m$, each with some secret data, and one computer C. For simplicity, we assume that C retains the answer.

We modify the basic protocol to achieve the same protection for $m + 1$ players that the basic protocol achieves for two, under the assumption that D_1 will not collude with C and that $D_2, \ldots, D_m$ can each send one message to C that D_1 will not intercept. D_1 generates k and shares it with everyone (including C), but does not reveal the factorization. Then each D_i produces encrypted bits and sends them to C, and C interacts with D_1 according to the

basic two-player protocol. This modification makes all data secure from everyone, under our assumption. In particular, C's questions do not reveal any secret data. In fact, the data is hidden from D_1 during the computation.

There is also a simple solution to the multiparty problem that does not rely on the assumption that D_1 will not collude with C and that $D_2, \ldots, D_m$ can each send one message to C that D_1 will not intercept. In this solution, C follows the two-player protocol with $D_1, \ldots, D_m$ as if they were a single player D. Together, $D_1, \ldots, D_m$ carry out all of the computations expected of D without revealing data to one another. For instance, $D_1, \ldots, D_m$ cooperate to generate k without giving any one of them enough information to factor k efficiently. Among themselves, $D_1, \ldots, D_m$ may use our protocol recursively, or they may use an equivalent protocol for multiparty computations with secret data.

6. On cheating

In this section, we address the question of what happens when either C or D tries to cheat. It is common to provide mechanisms to avoid or to detect cheating in protocols for secure computation, e.g., to turn protocols into "validated protocols" where the players use zero-knowledge subprotocols to prove that they have acted honestly ([GHY]). However, the issue of cheating is different in a setting in which some secrets are hidden rather than encrypted.

How might C try to cheat? He could try to choose the sequence of AND computations that he requests so as to compromise D's data rather than to compute $f(x)$. However, under the QRA, everything that she sends him during the gate-simulation phase of the protocol is something that he could generate himself – quadruples of elements of $Z_k^*[+1]$, three of which are nonresidues and one of which is a residue. In Case 1 of the final phase, she "opens" one bit, allowing him to learn $f(x)$. Thus, by using a different circuit, he could obtain a different boolean function of x, but nothing else. Because Case 1 of the final phase involves the opening of one bit, C could, instead, cheat by obtaining the value $r(u, k)$ for one arbitrary u.

It is meaningless to require that C prove that he has used the right circuit, because he is not committed to one circuit. In fact, he could not possibly be committed to a circuit, because commitment implies information transfer, and the protocol transfers no information to D about C's secrets. Similarly, it is impossible to require that C prove that he has requested the right bit without revealing some information. At the cost of some transfer of information about his circuit, C can be required to give zero-knowledge proofs that he has acted as required by the protocol (e.g., [BC1, GMW86]).

How might D try to cheat? We cannot accuse D of sending incorrect encrypted bits, because x is D's private data. D could deviate from the protocol by sending wrong answers when C asks her for the AND of two encrypted bits or by sending the wrong residuosity bit in Case 1 of the final phase. This could cause C to compute a wrong answer, but it would not compromise the privacy of C's secrets. C is capable of hiding; that is, none of the messages that he sends to D in the course of the computation conveys any information at all, even when D leads the computation astray. Therefore, D cannot cheat to "decrypt" C's messages and

learn something she is not supposed to know.

In any case, D can be required to prove that she computes the (encrypted) output of the AND gates correctly and that she sends the correct residuosity bit in the final phase. The set of tuples $\langle k,\ Y(b),\ Y(b'),\ Y(b_3),\ Y(b_2),\ Y(b_1),\ Y(b_0)\rangle$ that satisfy the following conditions is an NP set: $k = p \cdot q$; $Y(b)$ and $Y(b')$ are legal encryptions of bits; and $\langle Y(b_3),\ Y(b_2),\ Y(b_1),\ Y(b_0)\rangle$ is a legal reply from D to C's query about the AND of b and b'. Therefore, D can prove that her replies are correct with a zero-knowledge subprotocol. Similarly, in the final phase, we can require D to prove that she knows a certificate of the residuosity of $Y(f(x))$; that is, if $b = 1$, D must prove that she knows a number u such that $u^2 \equiv E(Y(f(x)))$ mod k, and, if $b = 0$, she must prove that she knows a number u such that $(-1) \cdot u^2 \equiv E(Y(f(x)))$ mod k.

7. Related work

There has been other recent work on secure circuit evaluation and related topics. Yao was the first to devise protocols for multiparty computations with secret inputs ([Y82, Y86]). His scheme involves the generation of many large primes during each run of the protocol and achieves encryption of secrets but not hiding. In independent work, Chaum *et al.*, Galil *et al.*, and Goldreich *et al.* developed new schemes in which some secrets are hidden (during some phases of the protocols) and in which the number of large primes required is reduced ([CDvdG, GHY, GMW87, GV]). All of these protocols conceal data, but not circuits. Furthermore, they are quite complex and their analyses are long and elusive.

The protocol that we present here is also related to our previous work with Kilian on hiding information from an oracle. In [AFK], we considered the scenario in which player A wishes to know $f(x)$ but lacks the computational resources to compute f. We asked whether she could query an oracle B, who was assumed to have unlimited computing power, and obtain $f(x)$ while hiding x. We provided some examples of functions f for which this is possible; one of the encryption schemes given in [AFK] is used as a building block in the basic protocol given here. However, the main result of [AFK] was negative: A cannot obtain $f(x)$ from B while revealing only $|x|$ for any f that is NP-hard. In this paper, we consider a more popular model, namely one in which the computation of f requires knowledge of secret algorithms or data, rather than oracular power. Here, our main result is positive.

8. An open problem

Our protocol uses the quadratic residuosity predicate $r(u, k)$ as a building block. No communication is required for C to compute the negation of an encrypted bit, but communication is required for C to compute a conjunction. Can C, by computing some function of $Y(b)$ and $Y(b')$, take the AND of two encrypted bits without asking for help from D? Brassard and Crépeau raised a similar question in [BC2], where they used a circuit-evaluation protocol based

on "permuted truth tables" in their work on zero-knowledge proof systems.

The quadratic residuosity predicate is not the only one that can be used as a building block in a protocol for secure circuit evaluation. To be usable as a building block, a boolean function must be encryptable (in the sense of [AFK]), be computable by player D (perhaps with knowledge of a secret key), and not be computable by player C. Is it possible to design a better protocol using a different building block? More specifically, we would like to reduce the number of rounds of communication needed by each run of the protocol, or, alternatively, to prove a nontrivial lower bound on the number of rounds needed for secure circuit evaluation.

9. Acknowledgements

We are grateful to Gilles Brassard for pointing out a bug in an earlier version and to Cynthia Hibbard for editorial help.

10. References

[AFK] Martín Abadi, Joan Feigenbaum, and Joe Kilian. "On Hiding Information from an Oracle," Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 19, 1987, 195–203.

[BC1] Gilles Brassard and Claude Crépeau. "Nontransitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond," Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 27, 1986, 188–195.

[BC2] Gilles Brassard and Claude Crépeau. "Zero-knowledge Simulation of Boolean Circuits," Advances in Cryptology — CRYPTO '86 Proceedings, Andrew Odlyzko (ed.), Springer-Verlag (pub.), 1987, 223–233.

[CDvdG] David Chaum, Ivan Damgaard, and Jeroen van de Graaf. "Multiparty Computations Ensuring Secrecy of Each Party's Input and Correctness of the Output," to appear in Proceedings of CRYPTO '87.

[GHY] Zvi Galil, Stuart Haber, and Moti Yung. "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model," to appear in Proceedings of CRYPTO '87.

[GM] Shafi Goldwasser and Silvio Micali. "Probabilistic encryption," *JCSS*, 28, 1984, 270–299.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design," Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 27, 1986, 174–187.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play ANY Mental Game," Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 19, 1987, 218–229.

[GV] Oded Goldreich and Ronen Vainish. "How to Solve any Protocol Problem — An Efficiency Improvement," to appear in Proceedings of CRYPTO '87.

[Y82] Andrew C. Yao. "Protocols for Secure Computations," Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 23, 1982, 160–164.

[Y86] Andrew C. Yao. "How to Generate and Exchange Secrets," Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 27, 1986, 162–167.