7

Alternative Threat Models

In this chapter, we consider some different assumptions about threats that lead to MPC protocols offering appealing security-performance trade-offs. First, we relax the assumption that any number of participants may be dishonest and discuss protocols designed to provide security only when a majority of the participants behave honestly. Assuming an honest majority allows for dramatic performance improvements. Then, we consider alternatives to the semi-honest and malicious models that have become standard in MPC literature, while still assuming that any number of participants may be corrupted. As discussed in the previous chapter, semi-honest protocols can be elevated into the malicious model, but this transformation incurs a significant cost overhead which may not be acceptable in practice. At the same time, real applications present a far more nuanced set of performance and security constraints. This prompted research into security models that offer richer trade-offs between security and performance. Section 7.1 discusses protocols designed to take advantage of the assumption that the majority of participants are honest. Section 7.2 discusses scenarios there trust between the participants is asymmetric, and the remaining sections present protocols designed to provide attractive security-performance trade-offs in settings motivated by practical scenarios.

7.1 Honest Majority

So far we have considered security against adversaries who may corrupt any number of the participants. Since the purpose of security is to protect the honest parties, the worst-case scenario for a protocol is that $n - 1$ out of n parties are corrupted.[1] In the two-party case, it is indeed the only sensible choice to consider one out of the two parties to be corrupt.

However, in the multi-party setting it often is reasonable to consider restricted adversaries that cannot corrupt as many parties as they want. A natural threshold is *honest majority*, where the adversary may corrupt strictly less than $\frac{n}{2}$ of the n parties. One reason this threshold is natural is that, assuming an honest majority, *every* function has an information-theoretically secure protocol (Ben-Or *et al.*, 1988; Chaum *et al.*, 1988), while there exist functions with no such protocol in the presence of $\lceil n/2 \rceil$ corrupt parties.

7.1.1 Building on Garbled Circuits

Yao's protocol (Section 3.1) provides semi-honest security, but to make it secure against malicious adversaries requires significant modifications with high computation and communication overhead (Section 6.1).

Mohassel *et al.* (2015) propose a simple *3-party variant* of Yao's protocol that is secure against a malicious adversary that corrupts at most one party (that is, the protocol is secure assuming an honest majority). Recall that the main challenge in making Yao's protocol secure against malicious attacks is to ensure that the garbled circuit is generated correctly. In the protocol of Mohassel *et al.* (2015), the main idea is to let two parties $\mathsf{P}_1, \mathsf{P}_2$ play the role of circuit garbler and one party P_3 play the role of circuit evaluator. First, P_1 and P_2 agree on randomness to be used in garbling, then they both garble the designated circuit *with the same randomness*, and send it to P_3. Since at most one of the garblers is corrupt (by the honest majority assumption), at least one of the garblers is guaranteed to be honest. Therefore, P_3 only needs to check that both garblers send identical garbled circuits. This ensures that the (unique)

[1] We consider only *static* security, where the adversary's choice of the corrupted parties is made once-and-for-all, at the beginning of the interaction. It is also possible to consider *adaptive* security, where parties can become corrupted throughout the protocol's execution. In the adpative setting, it does indeed make sense to consider scenarios where all parties are (eventually) corrupted.

garbled circuit is generated correctly. Other protocol issues relevant for the malicious case (like obtaining garbled inputs) are handled in a similar way, by checking responses from the two garbling parties for consistency.

One additional advantage of the 3-party setting is that there is no need for oblivious transfer (OT) as in the 2-party setting. Instead of using OT to deliver garbled inputs to the evaluator P_3, we can let P_3 secret-share its input and send one share (in the clear!) to each of P_1 and P_2. These garblers can send garbled inputs for each of these shares, and the circuit can be modified to reconstruct these shares before running the desired computation. Some care is required to ensure that the garblers send the correct garbled inputs in this way (Mohassel *et al.* (2015) provide details). Overall, the result is a protocol that avoids all OT and thus uses only inexpensive symmetric-key cryptography.

The basic protocol of Mohassel *et al.* (2015) has been generalized to provide additional properties like fairness (if the adversary learns the output, then the honest parties do) and guaranteed output delivery (all honest parties will receive output) (Patra and Ravi, 2018). Chandran *et al.* (2017) extend it to provide security against roughly $\sqrt{n}$ out of n corrupt parties.

7.1.2 Three-Party Secret Sharing

The honest-majority 3-party setting also enables some of the fastest general-purpose MPC implementations to date. These protocols achieve their high performance due to their extremely low communication costs — in some cases, as little as one bit per gate of the circuit!

It is possible to securely realize any functionality information-theoretically in the presence of an honest majority, using the classical protocols of Ben-Or *et al.* (1988) and Chaum *et al.* (1988). In these protocols, every wire in the circuit holds a value v, and the invariant of these protocols is that the parties collectively hold some additive secret sharing of v. As in Section 3.4, let $[v]$ denote such a sharing of v. For an addition gate $z = x + y$, the parties can compute a sharing $[x + y]$ from sharings $[x]$ and $[y]$ by local computation only, due to the additive homomorphism property of the sharing scheme. However, interaction and communication are required for multiplication gates to compute a sharing $[xy]$ from sharings $[x]$ and $[y]$.

In Section 3.4 we discussed how to perform such multiplications when pre-processed triples of the form $[a], [b], [ab]$ are available. It is also possible

to perform multiplications with all effort taking place during the protocol (i.e., not in a pre-processing phase). For example, the protocol of Ben-Or *et al.* (1988) uses Shamir secret sharing for its sharings $[v]$, and uses an interactive multiplication subprotocol in which all parties generate shares-of-shares and combine them linearly.

The protocols in this section are instances of this general paradigm, highly specialized for the case of 3 parties and 1 corruption ("1-out-of-3" setting). Both the method of sharing and the corresponding secure multiplication subprotocol are the target of considerable optimizations.

The Sharemind protocol of Bogdanov *et al.* (2008a) was the first to demonstrate high performance in this setting. Generally speaking, for the 1-out-of-3 case, it is possible to use a secret sharing scheme with threshold 2 (so that any 2 shares determine the secret). The Sharemind protocol instead uses 3-out-of-3 additive sharing, so that in $[v]$ party P_i holds value v_i such that $v = v_1 + v_2 + v_3$ (in an appropriate ring, such as $\mathbb{Z}_2$ for Boolean circuits). This choice leads to a simpler multiplication subprotocol in which each party sends 7 ring elements.

Launchbury *et al.* (2012) describe an alternative approach in which each party sends only 3 ring elements per multiplication. Furthermore, the communication is in a round-robin pattern, where the only communication is in the directions $\mathsf{P}_1 \to \mathsf{P}_2 \to \mathsf{P}_3 \to \mathsf{P}_1$. The idea behind multiplication is as follows. Suppose two values x and y are additively shared as $x = x_1 + x_2 + x_3$ and $y = y_1 + y_2 + y_3$, where party P_i holds x_i, y_i. To multiply x and y, it suffices to compute all terms of the form $x_a \cdot y_b$ for $a, b \in \{1, 2, 3\}$. Already P_i has enough information to compute $x_i y_i$, but the other terms are problematic. However, if each P_i sends its shares around the circle (i.e., P_1 sends to P_2, P_2 sends to P_3, and P_3 to P_1), then every term of the form $x_a y_b$ will computable by some party. Each party will now hold two of the x_i shares and two of the y_i shares. This still perfectly hides the values of x and y from a single corrupt party since we are using 3-out-of-3 sharing. The only problem is that shares of xy are correlated to shares of x and y, while it is necessary to have an independent sharing of xy. So, the parties generate a random additive sharing of zero and locally add it to their (non-random) sharing of xy.

Araki *et al.* (2016) propose a secret sharing scheme that is a variant of

replicated secret sharing. The sharing of a value v is defined as:

$$\mathsf{P}_1 \text{ holds } (x_1, x_3 - v) \qquad \mathsf{P}_2 \text{ holds } (x_2, x_1 - v) \qquad \mathsf{P}_3 \text{ holds } (x_3, x_2 - v)$$

where the x_i values are a random sharing of zero: $0 = x_1 + x_2 + x_3$. They describe a multiplication subprotocol in which each party sends only one ring element (in a round-robin fashion as above).[2] One of the ways in which communication is minimized in this approach is that the multiplication subprotocol generates its needed randomness *without any interaction.* Suppose there are three keys, k_1, k_2, k_3, for a pseudorandom function where P_1 holds (k_1, k_2), P_2 holds (k_2, k_3), and P_3 holds (k_3, k_1). Then, the parties can non-interactively generate an unbounded number of random sharings of zero. The i-th sharing of zero is generated via:

- P_1 computes $s_1 = F_{k_1}(i) - F_{k_2}(i)$
- P_2 computes $s_2 = F_{k_2}(i) - F_{k_3}(i)$
- P_3 computes $s_3 = F_{k_3}(i) - F_{k_1}(i)$

Here F is a pseudorandom function whose output is in the field. As desired, $s_1 + s_2 + s_3 = 0$ and the sharing is indeed random from the perspective of a single corrupt party. Such random sharings of zero are the only randomness needed by the multiplication subprotocol of Araki *et al.* (2016).

As a result of such minimal communication, Araki *et al.* (2016) report MPC performance of over 7 billion gates per second. This performance suffices to replace a single Kerberos authentication server with 3 servers running MPC, so that no single server ever sees user passwords, and is able to support loads of 35,000 logins per second.

These results are secure against one *semi-honest* corrupt party. Furukawa *et al.* (2017) extend the approach of Araki *et al.* (2016), and show how to achieve security against one *malicious* party (for the case of Boolean circuits). Their protocol follows the high-level approach described in Section 3.4, by generating multiplication triples $[a], [b], [ab]$. These triples use the secret-sharing technique of Araki *et al.* (2016). The starting point to generate such triples is also the multiplication protocol of Araki *et al.* (2016). Here, an

[2] Their protocol works as long as the number 3 is invertible in the ring. In particular, it can be used for Boolean arithmetic over $\mathbb{Z}_2$ as well as $\mathbb{Z}_{2^k}$.

important property of this protocol is used: a single corrupt party cannot cause the output of multiplication to be an *invalid* sharing, only a (valid) sharing of a different value. Hence, the adversary can cause any triple generated in this way to have the form $[a], [b], [ab]$ or $[a], [b], [\overline{ab}]$ (since this idea only applies for sharings of single bits). Starting with this observation, collections of triples are used to "cross-check" each other and guarantee their correctness.

7.2 Asymmetric Trust

Although the standard models assume all parties are equally distrusting, many realistic scenarios have asymmetric trust. For example, consider the setting where one of the two participants of a computation is a well-known business, such as a bank (denoted by P_1), providing service to another, less trusted, participant, bank's customer, denoted by P_2. It may be safe to assume that P_1 is unlikely to actively engage in cheating by deviating from the prescribed protocol. Indeed, banks today enjoy full customer trust and operate on all customer data in plaintext. Customers are willing to rely on established regulatory and legal systems, as well as the long-term reputation of the bank, rather than on cryptographic mechanisms, to protect their funds and transactions. Today, we not only trust the bank to correctly perform requested transactions, but we also trust that the bank will not misuse our data and will keep it in confidence.

However, there may be several reasons why a cautious customer who trusts the bank's intent may want to withhold certain private information and execute transactions via MPC. One is the *unintentional* data release. As with any organization, the bank may be a target of cyber attacks, and data stored by the bank, including customer data, may simply be stolen. Having employed MPC to safeguard the data eliminates this possibility, since the bank will not have the sensitive data in the first place. Another reason might be the legally mandated audits and summons of the data. As an organization, a bank may have a presence in several jurisdictions with different rules on data retention, release and reporting. Again, MPC will serve as a protection for unpredictable future data releases.

Hence, given the existing trust to the bank it seems reasonable to employ semi-honest model to protect the customer. However, having upgraded customer privacy by moving from plaintext operation to semi-honest MPC (and correspondingly placing the customer as a semi-honest player), we now actually

greatly compromised bank's security. Indeed, where previously the customer has been a passive participant simply providing the input to the transaction, now the customer has opportunities to cheat within the semi-honest MPC protocol. Given the relative ease of creating an account at a bank, and the opportunity for financial gain by an improperly conducted financial transaction, a bank's customer is naturally incentivized to cheat.

This scenario suggests a hybrid model where one player is assumed to be semi-honest and the other is assumed to be malicious. Luckily, Yao's GC and many of its natural variants are *already secure against malicious evaluator*! Indeed, assuming the OT protocol used by P_2 to obtain its input wires provides security against a malicious evaluator, P_2, the GC evaluator proceeds simply by decrypting garbled tables in a single, pre-determined way. Any deviation by the GC evaluator in the GC evaluation will simply result in a failure to obtain an output label. This is a very convenient feature of Yao's GC, which allows for an order of magnitude or more cost savings in settings where the security policy can assume a semi-honest generator (P_1).

Server-aided secure computation is another noteworthy example of taking advantage of asymmetric trust. The Salus system (Kamara *et al.*, 2012) considers a natural setting where one participant is a special external party without input or output, whose goal is to assist the other parties in securely evaluating the function on their inputs. The external party might be a cloud service provider with larger computational resources at its disposal and, importantly, sufficient status to be partially trusted by all the other participants. For example, this player may be trusted to behave semi-honestly, while other players may be assumed to be malicious. An important assumption made by Kamara *et al.* (2012) is that the server *does not collude* with any of the other players, even if the server is malicious. Substantial performance improvements can be obtained by the Salus systems and subsequent work by taking advantage of the asymmetric trust and the collusion restriction. For example, Kamara *et al.* (2014) present a server-aided private set intersection protocol, which, in the case of the semi-honest server, computes PSI of billion-element sets in about 580 seconds while sending about 12.4 GB of data.

7.3 Covert Security

While reasonable in many settings, at times the above semi-honest/malicious asymmetric trust model is insufficient for applications where P_1 may have incentives to cheat. Even for a mostly trusted bank, cheating is nevertheless a real possibility, and bank customers and even external auditors have no tools to ensure compliance. At the same time, moving fully into the malicious model will incur a large performance penalty that may not be acceptable.

An effective practical approach may be to enable probabilistic checks on the generator, as first proposed by Aumann and Lindell (2007). Their idea is to allow the evaluator (P_2) to challenge the generator (P_1) at random to prove that the garbled circuit was constructed correctly. If P_1 is unable to prove this, then P_2 will know that P_1 is cheating and will react accordingly. The guarantee achieved is that a cheating player will be caught with a certain fixed probability (e.g., $\epsilon = \frac{1}{2}$) known as the *deterrence factor*.

This simple and natural idea can be formalized in the definition in several ways. First, it is essential that P_1 cannot forge an invalid proof or elude or withdraw from a validity proof challenge on a garbled circuit it produced. On the other hand, we accept that if P_2 did not challenge P_1 on an improperly-generated circuit, then P_1 was not caught cheating, and can win (i.e., learn something about P_2's private input or corrupt the function output). However, various privacy guarantees with respect to P_2's input may be considered. Aumann and Lindell (2007) propose three formulations:

1. *Failed simulation*. The idea is to allow the simulator (of the cheating party) to fail sometimes. "Fail" means that its output distribution is not indistinguishable from the real one. This corresponds to an event of successful cheating. The model guarantees that the probability that the adversary is caught cheating is at least ϵ times the probability that the simulator fails.

   One serious issue with the above definition is that it *only* requires that if cheating occurred in the real execution, the cheater will be caught with probability ϵ. The definition does not prevent the cheating player from deciding *when* to cheat (implicitly) based on the *honest* player's input. In particular, P_1 could attempt cheat only on the more valuable inputs of P_2 (e.g., natural protocols exist which allow P_1 to attempt to cheat

only when P_2's first bit is 0).

2. *Explicit cheat*. This formulation introduces an explicit ability to cheat to an ideal-model adversary (i.e., the simulator). This only slightly complicates the definition, but allows us to naturally prescribe that cheating in the ideal model can only occur *independently* of the other players' inputs. In the ideal model, the cheating player, upon sending a cheat instruction to the trusted party, will obtain the honest players' inputs. At the same time, the honest players in the ideal model will output $\mathsf{corrupted}_i$ (i.e., detect cheating by P_i) with probability ϵ, thus requiring that the same happens in the real model.

   Although this model is much more convincing, it has the drawback that the malicious player is allowed to obtain inputs of the honest parties even when cheating is detected. As Aumann and Lindell noted, "there is less deterrence to not rob a bank if when you are caught you are allowed to keep the stolen money." (Aumann and Lindell, 2007)

3. *Strong explicit cheat*. This is the same as the explicit cheat formulation, with the exception that the cheating ideal-model adversary *is not allowed* to obtain the honest players' inputs in the case where cheating is detected.

The first two (strictly weaker) models of Aumann and Lindell (2007) did not gain significant popularity mainly because the much stronger third model admits protocols of the same or very similar efficiency as the weaker ones. The *strong explicit cheat* model became standard due to its simplicity, effectiveness in deterrence, and the discovery of simple and efficient protocols that achieve it. We present one such simple and efficient 2PC protocol next.

7.3.1 Covert Two-Party Protocol

Since the work of Aumann and Lindell (2007), significant progress in efficient OT has produced several extremely efficient *malicious* OT protocols (Asharov *et al.*, 2015b; Keller *et al.*, 2015), with the latter having overhead over the semi-honest OT of only 5%. As a result, we don't consider covert OT security, and assume a maliciously-secure OT building block. It is important to remember, however, that a maliciously-secure protocol does not guarantee the players submit prescribed inputs. In particular, while malicious OT ensures correct

and private OT execution, deviating players can submit arbitrary OT inputs, such as invalid wire labels.

Next, we overview the method of Aumann and Lindell (2007) for building a covert 2PC protocol from Yao's GC and malicious OT. For simplicity, we assume deterrence factor of $\epsilon = \frac{1}{2}$. It is straightforward to efficiently generalize this for any non-negligible ϵ. First, we present the basic idea and point out missing pieces, which we then address.

Core Protocol. Aumann and Lindell go along the lines of the cut-and-choose approach and propose that P_1 generates and sends to P_2 *two* GCs. The two garbled circuits $\widehat{C}_0$ and $\widehat{C}_1$ are generated from random seeds s_0 and s_1 respectively by expanding them using a pseudo-random generator (PRG). Upon receipt, P_2 flips a coin $b \in \{0, 1\}$ and asks P_1 to *open* the circuit $\widehat{C}_b$ by providing s_b. Because the GCs are constructed deterministically from a seed via a PRG expansion, opening is achieved simply by sending the seed to the verifier. This allows P_2 to check the correctness of the generated garbled circuit $\widehat{C}_b$ by constructing the same garbled circuit from the provided seed, and comparing it to the copy that was sent. This guarantees that a badly constructed $\widehat{C}$ will be detected with probability $\epsilon = \frac{1}{2}$, which is needed to satisfy the strong explicit cheat definition.

However, a malicious P_1 can also perform OT-related attacks. For example, P_1 can flip the semantics of the labels on P_2's input wires, effectively silently flipping P_2's input. Similarly, P_1 can set *both* P_2's input wire labels to the same value, effectively setting P_2's input to a fixed value. Another attack is the selective abort attack discussed in Section 6.1, where one of the two OT secrets is set to be a dummy random value, resulting in an selectively aborted evaluation that allows P_1 to learn a bit of P_2's input.

As a result, we must ensure that a OT input substitution by P_1 is caught at least with probability equal to the deterrence factor ϵ. Note that input substitution by P_2 is allowed as it simply corresponds to P_2 choosing a different MPC input, a behavior allowed by the security definition.

Next, we discuss defenses to these attacks.

Preventing OT Input Substitution and Selective Abort. The solution enhances the basic protocol described above to provide additional protections

against OT input substitution by P_1. First, recall that the *entire* GC is generated from a seed, which includes the input wire labels. Further, OT is run prior to P_2's challenge being announced, so P_1 will not be able to adjust its tactic in response to the challenge and provide honest OT inputs in the challenge execution and malicious OT inputs in the live execution. At the same time, OT delivers only one of the two inputs to P_2, and P_2 cannot verify that the other OT input was correctly submitted by P_1, leaving open the possibility of the selective abort attack, described above.

This is not satisfactory in the strong explicit cheat formulation, since this model requires that if cheating is detected, the dishonest party cannot learn anything about the honest player's input. There are several ways to address this. Aumann and Lindell suggest using the inputs XOR tree idea, which was earlier proposed by Lindell and Pinkas (2007). The idea is to modify the circuit C being computed and correspondingly change the semantics of P_2's input wires as follows. Instead of each of P_2's input bit x_i, the new circuit C' will have σ inputs $x_i^1, ..., x_i^\sigma$, which are random with the restriction that $x_i = \bigoplus_{j \in \{1..\sigma\}} x_i^j$. For each x_i of C, the new circuit C' will start by xor-ing the σ inputs $x_i^1, ..., x_i^\sigma$ so as to recover x_i inside the circuit. C' then continues as the original C would with the reconstructed inputs. The new circuit computes the same function, but now P_1 can only learn information if it is correctly guesses *all* of the σ random x_i^j values, an event occurring with statistically negligible probability.

Alternate Keys. We additionally mention the following mainly theoretical attack discussed by Aumann and Lindell (2007) to underline the subtleties of even seemingly simple MPC protocols. The issue is that an adversary can construct (at least in theory) a garbled circuit with two sets of keys, where one set of keys decrypts the circuit to the specified one and another set of keys decrypts the circuit to an incorrect one. This is not a real issue in most natural GC constructions, but one can construct a tailored GC protocol with this property. The existence of two sets of keys is problematic because the adversary can supply "correct keys" to the circuits that are opened and "incorrect keys" to the circuit that is evaluated. Aumann and Lindell prevent this by having P_1 commit to these keys and send the commitments together with the garbled circuits. Then, instead of P_1 just sending the keys associated with its input, it sends the appropriate decommitments.

Protocol Completion. Finally, to conclude the informal description of the protocol, after P_2 successfully conducts the above checks, it proceeds by evaluating $\widehat{C}_{1-b}$ and obtaining the output which is then also sent to P_1. It now can be shown that malicious behavior of P_1 can be caught with probability $\epsilon = \frac{1}{2}$. We note that this probability can be increased simply by having P_1 generate and send more circuits, all but one of which are opened and checked. Aumann and Lindell (2007) provide a detailed treatment of the definitions and the formal protocol construction.

7.4 Publicly Verifiable Covert (PVC) Security

In the covert security model, a party can deviate arbitrarily from the protocol description but is caught with a fixed probability ϵ, called the *deterrence factor*. In many practical scenarios, this guaranteed risk of being caught (likely resulting in loss of business or embarrassment) is sufficient to deter would-be cheaters, and covert protocols are much more efficient and simpler than their malicious counterparts.

At the same time, the cheating deterrent introduced by the covert model is relatively weak. Indeed, an honest party catching a cheater certainly knows what happened and can respond accordingly (e.g., by taking their business elsewhere). However, the impact is largely limited to this, since the honest player cannot credibly *accuse* the cheater publicly. Doing so might require the honest player to reveal its private inputs (hence, violate its security), or the protocol may simply not authenticate messages as coming from a specific party. If, however, credible public accusation (i.e., a publicly-verifiable cryptographic proof of the cheating) were possible, the deterrent for the cheater would be much greater: suddenly, *all* the cheater's customers and regulators would be aware of the cheating and thus any cheating may affect the cheater's global customer base.

The addition of credible accusation greatly improves the covert model even in scenarios with a small number of players, such as those involving the government. Consider, for example, the setting where two agencies are engaged in secure computation on their respective classified data. The covert model may often be insufficient here. Indeed, consider the case where one of the two players deviates from the protocol, perhaps due to an insider attack. The honest player detects this, but non-participants are now faced with the problem

of identifying the culprit across two domains, where the communication is greatly restricted due to trust, policy, data privacy legislation, or all of the above. On the other hand, credible accusation immediately provides the ability to exclude the honest player from the suspect list, and focus on tracking the problem within the misbehaving organization, which is dramatically simpler.

PVC Definition. Asharov and Orlandi (2012) proposed a security model, *covert with public verifiability*, and an associated protocol, motivated by these concerns. At a high level, they proposed that when cheating is detected, the honest player can publish a "certificate of cheating" that can be checked by any third party. We will call this model PVC, following the notation of Kolesnikov and Malozemoff (2015), who proposed an improved protocol in this model.

Informally, the PVC definition requires the following three properties to hold with overwhelming probability:

1. Whenever cheating is detected, the honest party can produce a publicly verifiable proof of cheating.

2. Proof of cheating cannot be forged. That is, an honest player cannot be accused of cheating with a proof that verifies.

3. Proof of cheating does not reveal honest party's private data (including the data used in the execution where cheating occurred).

Asharov-Orlandi PVC Protocol. The Asharov-Orlandi protocol has performance similar to the original covert protocol of Aumann and Lindell (2007) on which it is based, with the exception of requiring signed-OT, a special form of oblivious transfer (OT). Their signed-OT construction, which we summarize next, is based on the OT of Peikert *et al.* (2008), and thus requires several expensive public-key operations per OT instance. After this, we describe several performance improvements to it proposed by Kolesnikov and Malozemoff (2015), the most important of which is a novel signed-OT extension protocol that eliminates per-instance public-key operations.

As usual, we make P_1 the circuit generator, P_2 the evaluator, and use C to represent the circuit to execute. Recall from Section 7.2 that in the standard Yao's garbled circuit construction in the semi-honest model, a malicious evaluator (P_2) cannot cheat during the GC evaluation. Hence, this protection

comes for free with natural GC protocols, and we only need to consider a malicious generator (P_1).

Recall the selective failure attack on P_2's input wires, where P_1 sends P_2 (via OT) an invalid wire label for one of P_2's two possible inputs and learns which input bit P_2 selected based on whether or not P_2 aborts. To protect against this attack, the parties construct a new circuit C' that prepends an input XOR tree in C as discussed in Section 7.3. To elevate to the covert model, P_1 then constructs λ (the *GC replication factor*) garblings of C' and P_2 randomly selects $\lambda - 1$ of them and checks they are correctly constructed, and evaluates the remaining C' garbled circuit to derive the output.

We now adapt this protocol to the PVC setting by allowing P_2 to not only detect cheating, but also to obtain a publicly verifiable proof of cheating if cheating is detected. The basic idea is to require the generator P_1 to establish a public-private keypair, and to *sign* the messages it sends. The intent is that signed inconsistent messages (e.g., badly formed GCs) can be published and will serve as a convincing proof of cheating. The main difficulty of this approach is ensuring that neither party can improve its odds by selectively aborting. For example, if P_1 could abort whenever P_2's challenge would reveal that P_1 is cheating (and hence avoid sending a signed inconsistent transcript), this would enable P_1 to cheat without the risk of generating a proof of cheating.

Asharov and Orlandi address this by preventing P_1 from knowing P_2's challenge when producing the response. In their protocol, P_1 sends the GCs to P_2 and opens the checked circuits by responding to the challenge through a 1-out-of-λ OT. For this, P_1 first sends all (signed) GCs to P_2. Then the players run OT, where in the i-th input to the OT P_1 provides openings (seeds) for all the GCs except for the i-th, as well as the input wire labels needed to evaluate $\widehat{C}_i$. Party P_2 inputs a random $\gamma \in_R [\lambda]$, so receives the seeds for all circuits other than $\widehat{C}_\gamma$ from the OT and the wire labels for its input for $\widehat{C}_\gamma$. Then, P_2 checks that all GCs besides $\widehat{C}_\gamma$ are constructed correctly; if the check passes, P_2 evaluates $\widehat{C}_\gamma$. Thus, P_1 does not know which GC is being evaluated, and which ones are checked.

However, a more careful examination shows that this actually does not quite get us to the PVC goal. Indeed, a malicious P_1 simply can include invalid openings for the OT secrets which correspond to the undesirable choice of the challenge γ. The Asharov-Orlandi protocol addresses this by having P_1

sign all its messages as well as using a *signed*-OT in place of all standard OTs (including wire label transfers and GC openings). Informally, the signed-OT functionality proceeds as follows. Rather than the receiver $\mathcal{R}$ getting message $\mathbf{m}_b$ (which might include a signature that $\mathcal{S}$ produced) from the sender $\mathcal{S}$ for choice bit b, the signature component of signed-OT is explicit in the OT definition. Namely, we require that $\mathcal{R}$ receives $((b, \mathbf{m}_b), \mathsf{Sig})$, where Sig is $\mathcal{S}$'s valid signature of $(b, \mathbf{m}_b)$. This guarantees that $\mathcal{R}$ will always receive a valid signature on the OT output it receives. Thus, if $\mathcal{R}$'s challenge detects cheating, the (inconsistent) transcript will be signed by $\mathcal{S}$, so can be used as proof of this cheating. Asharov and Orlandi (2012) show that this construction is ϵ-PVC-secure for $\epsilon = (1 - 1/\lambda)(1 - 2^{-\nu+1})$, where ν is the replication factor of the employed XOR tree, discussed above.

We note that their signed-OT heavily relies on public-key operations, and cannot use the much more efficient OT extension.

Signed-OT Extension. Kolesnikov and Malozemoff (2015) proposed an efficient signed-OT extension protocol built on the malicious OT extension of Asharov *et al.* (2015b). Informally, signed-OT extension (similarly to the signed-OT of Asharov-Orlandi) ensures that (1) a cheating sender $\mathcal{S}$ is held accountable in the form of a "certificate of cheating" that the honest receiver $\mathcal{R}$ can generate, (2) the certificate of cheating *does not reveal* honest player's inputs and, (3) a malicious $\mathcal{R}$ *cannot defame* an honest $\mathcal{S}$ by fabricating a false "certificate of cheating".

Achieving the first goal is fairly straightforward and can be done by having $\mathcal{S}$ simply sign *all* its messages sent in the course of executing OT extension. Then, the view of $\mathcal{R}$, which will now include messages signed by $\mathcal{S}$, will exhibit inconsistency. The challenge is in simultaneously:

1. protecting the privacy of $\mathcal{R}$'s input—this is a concern since the view of $\mathcal{R}$ exhibiting inconsistency also may include $\mathcal{R}$'s input, and

2. preventing a malicious $\mathcal{R}$ from manipulating the part of the view which is not under $\mathcal{S}$'s signature to generate a false accusation of cheating.

Since the view of $\mathcal{R}$ plays the role of the proof of cheating, we must ensure certain non-malleability of the view of $\mathcal{R}$, to prevent it from defaming the honest $\mathcal{S}$. For this, we need to commit $\mathcal{R}$ to its particular choices throughout

the OT extension protocol. At the same time, we must maintain that those commitments do not leak any information about $\mathcal{R}$'s choices. Next, we sketch how this can be done, assuming familiarity with the details of the OT extension of Ishai *et al.* (2003) (IKNP from Section 3.7.2).

Recall that in the standard IKNP OT extension protocol, $\mathcal{R}$ constructs a random matrix M, and $\mathcal{S}$ obtains a matrix M' derived from the matrix M, $\mathcal{S}$'s random string $\mathbf{s}$, and $\mathcal{R}$'s vector of OT inputs $\mathbf{r}$. The matrix M is the main component of $\mathcal{R}$'s view which, together with $\mathcal{S}$'s signed messages will constitute a proof of cheating.

To reiterate, we must address two issues. First, because M' is obtained by applying $\mathcal{R}$'s private input $\mathbf{r}$ to M, and M' is known to $\mathcal{S}$, M is now sensitive and cannot be revealed. Second, we must prevent $\mathcal{R}$ from publishing a doctored M, which would enable a false cheating accusation. Kolesnikov and Malozemoff (2015) (KM) resolve both issues by observing that $\mathcal{S}$ does in fact learn some of the elements of M, since in the OT extension construction some of the columns of M and M' are the same (i.e., those corresponding to zero bits of $\mathcal{S}$'s string $\mathbf{s}$).

The KM signed-OT construction prevents $\mathcal{R}$ from cheating by having $\mathcal{S}$ include in its signature carefully selected information from the columns in M which $\mathcal{S}$ sees. Finally, the protocol requires that $\mathcal{R}$ generate each row of M from a seed, and that $\mathcal{R}$'s proof of cheating includes this seed such that the row rebuilt from the seed is consistent with the columns included in $\mathcal{S}$'s signature. Kolesnikov and Malozemoff (2015) show that this makes it infeasible for $\mathcal{R}$ to successfully present an invalid row of the OT matrix in the proof of cheating. The KM construction is in the random oracle model, a slight strengthening of the assumptions needed for standard OT extension and FreeXOR, two standard secure computation tools.

The KM construction is also interesting from a theoretical perspective in that it shows how to construct signed-OT from *any* maliciously secure OT protocol, whereas Asharov and Orlandi (2012) build a specific construction based on the Decisional Diffie-Hellman problem.

7.5 Reducing Communication in Cut-and-Choose Protocols

A basic technique in cut-and-choose used to achieve malicious-level security, and in the covert and PVC models, is for P_1 to send several garbled circuits,

of which several are opened and checked, and one (or more for malicious security) is evaluated. The opened garbled circuits have no further use, since they hold no secrets. They only serve as a commitment for purposes of the challenge protocol. Can *commitments* to these GCs be sent and verified instead, achieving the same utility?

Indeed, as formalized by Goyal *et al.* (2008), this is possible in covert and malicious cut-and-choose protocols. One must, of course, be careful with the exact construction. One suitable construction is provided by Goyal *et al.* (2008). Kolesnikov and Malozemoff (2015) formalize a specific variant of hashing, which works with their PVC protocol, thus resulting of the PVC protocol being of the same communication cost as the semi-honest Yao's GC protocol.

Free Hash. While GC hashing allows for significant communication savings, it may not provide much overall savings in total execution time since hashing is a relatively expensive operation. Motivated by this, Fan *et al.* (2017) proposed a way to compute a GC hash simply by xor-ing (slightly malleated) entries of garbled tables. Their main idea is to *greatly weaken* the security definition of the GC hash. Instead of requiring that it is infeasible to find a hash collision, they *allow* an adversary to generate a garbled circuit $\widehat{C}'$ whose hash collides with an honestly generated $\widehat{C}$, as long as such a $\widehat{C}'$ with high probability will fail evaluation and cheating will be discovered. Fan *et al.* (2017) then show how to intertwine hash generation and verification with GC generation and evaluation, such that the resulting hash is generated at no extra computational cost and meets the above definition.

7.6 Trading Off Leakage for Efficiency

Maliciously secure MPC provides extremely strong security guarantees, at a cost. In many cases, the security needs may allow for less than absolute inability of a malicious attacker to learn even a single bit of information. At the same time, a large majority of the cost of MPC comes from the "last mile" of entirely protecting all private information. Given this, useful trade-offs between MPC security and efficiency have been explored. In this section we discuss the dual-execution approach of Mohassel and Franklin (2006), as well as several private database systems.

Dual Execution. The *dual-execution* 2PC protocol of Mohassel and Franklin (2006) capitalizes on the fact that only the circuit generator is able to cheat in 2PC GC, assuming malicious-secure OT. Indeed, the evaluator simply performs a sequence of decryptions, and deviation of the prescribed protocol results in the evaluator being stuck, which is equivalent to abort. The idea behind dual execution is to allow *both* players to play a role of the generator (hence forcing the opponent to play the role of the evaluator where it will not be able to cheat). As a result, in each honest player's view, the execution where it was the generator must be the correct one. However, the honest player additionally must ensure that the other execution is not detrimental to security, for example, by leaking private data.

To achieve this, Mohassel and Franklin (2006) propose that the two executions must produce the same candidate output—a difference in candidate outputs in the two executions implies cheating, and the computation must be aborted without output to avoid leaking information from an invalid circuit. In the protocol, the two parties run two separate instances of Yao's semi-honest protocol, so that for one instance P_1 is the generator and P_2 is the evaluator, and for the other instance P_2 is the generator and P_1 is the evaluator. Each party evaluates the garbled circuit they received to obtain a (still garbled) output. Then the two parties run a *fully maliciously secure* equality test protocol to check whether their outputs are semantically equal (before they are decoded). Each party inputs both the garbled output they evaluated and the output wire labels of the garbled circuit they generated. If the outputs don't match, then the parties abort. Of course, abort is an exception, and the aborting honest party will know that that other player cheated in the execution. While this equality check subprotocol is evaluated using a malicious-secure protocol, its cost is small since the computed comparison function is fixed and only depends on the size of the output.

The dual-execution protocol is *not* fully secure in the malicious model. Indeed, the honest party executes an adversarially-crafted garbled circuit and uses the garbled output in the equality-test subprotocol. A malicious party can generate a garbled circuit that produces the correct output for some of the other party's inputs, but an incorrect one for others. Then, the attacker learns whether the true function output is equal to an arbitrary predicate on honest party input by observing the abort behavior of the protocol. However, since the

equality test has only one bit of output, it can be shown that the dual-execution protocol leaks at most one (adversarially-chosen) bit describing the honest party's input.

In a follow-up work, Huang *et al.* (2012b) formalize the dual-execution definition of Mohassel and Franklin and propose several optimizations including showing how the two executions can be interleaved to minimize the overall latency of dual execution overhead over semi-honest single execution. In another follow-up work, Kolesnikov *et al.* (2015) show how the leakage function of the dual-execution 2PC can be greatly restricted and the probability of leakage occurring reduced.

Memory Access Leakage: Private DB Querying. Another strategy for trading-off some leakage for substantial efficiency gains is to incorporate custom protocols into an MPC protocol that leak some information for large performance gains. This can be particularly effective (and important) for applications involving large-scale data analysis. Large-scale data collection and use has become essential to operation of many fields, such as security, analysis, optimization, and others. Much of the data collected and analyzed is private. As a result, a natural question arises regarding the feasibility of MPC application to today's data sets.

One particular application of interest is private database querying. The goal is to enable a database server (DB) to respond to client's *encrypted* queries, so as to protect the sensitive information possibly contained in the query. Private queries is a special case of MPC, and can be instantiated by using generic MPC techniques.

One immediate constraint, critical to our discussion is the necessity of *sublinear* data access. A fully secure approach would involve scanning the *entire* DB for a single query to achieve semi-honest security. This is because omitting even a single DB entry reveals to the players that this entry was not in the result set. Sublinear execution immediately implies loss of security of MPC even in the semi-honest model. However, as covered in Chapter 5, when linear-time preprocessing is allowed, amortized costs of data access can be sublinear while achieving full formal cryptographic guarantees.

Known algorithms for fully secure sublinear access, however, are still very expensive, bringing 3–4 orders of magnitude performance penalties.

A promising research direction looks into allowing certain (hopefully, well-understood) leakage of information about private inputs in the pursuit of increased efficiency in protocols for large-scale data search.

Several systems designed to provide encrypted database functionalities use MPC as an underlying technology (Pappas *et al.*, 2014; Poddar *et al.*, 2016). In the Blind Seer project (Pappas *et al.*, 2014; Fisch *et al.*, 2015), two- and three-party MPC was used as an underlying primitive for implementing encrypted search tree traversal, performed jointly by the client and the server.

Both areas are active research areas, and several commercial systems are deployed that provide searchable encryption and encrypted databases. We note that today we don't have precise understanding of the impact of the leaked information (or the information contained in the authorized query results sets, for that matter). Several attacks have shown that data leaked for the sake of efficiency can often be exploited (Islam *et al.*, 2012; Naveed *et al.*, 2015; Cash *et al.*, 2015). Understanding how to make appropriate trade-offs between leakage and efficiency remains an important open problem, but one that many practical systems must face.

7.7 Further Reading

MPC today is already fast enough to be applied to many practical problems, yet larger-scale MPC applications are often still impractical, especially in the fully-malicious model. In this chapter, we discussed several approaches that aim to strike a balance between security and performance. Reducing the adversary power (i.e., moving from malicious to covert and PVC models) enables approximately a factor of 10 cost reduction compared to authenticated garbling (Section 6.7) and a factor of 40 compared to the best cut-and-choose methods (Section 6.1). Allowing for additional information to be revealed promises even more significant performance improvements, up to several orders of magnitude in some cases (as seen in the private database work in Section 7.6). Of course, it is important to understand the effect of the additional leakage. We stress, however, that there is no fundamental difference in the *extra* leakage during the protocol execution, and the allowed information obtained from the output of the computation. Both can be too damaging and both should be similarly analyzed to understand if (securely) computing the proposed function is safe.

Another approach that can produce dramatic performance improvements is employing secure hardware such as Intel SGX or secure smartcards (Ohrimenko *et al.*, 2016; Gupta *et al.*, 2016; Bahmani *et al.*, 2017; Zheng *et al.*, 2017). This offers a different kind of a trade-off: should we trust the manufacturer of the hardware (we must consider both competence and possible prior or future malicious intent) to greatly improve performance of secure computing? There is no clear answer here, as hardware security seems to be a cat-and-mouse game, with significant attacks steadily appearing, and manufacturers catching up in their revisions cycle. Examples of recent attacks on SGX include devastating software-only key recovery (Bulck *et al.*, 2018), which does not make any assumption on victim's enclave code and does not necessarily require kernel-level access. Other attacks include exploits of software vulnerabilities, e.g., (Lee *et al.*, 2017a), or side channels, e.g., (Xu *et al.*, 2015; Lee *et al.*, 2017b). Ultimately, delivering high-performance in a CPU requires very complex software and hardware designs, which are therefore likely to include subtle errors and vulnerabilities. Secure enclaves present an attractive and high-value attack target, while their vulnerabilities hard to detect in the design cycle. As such, they may be suitable for computing on lower-value data in larger instances where MPC is too slow, but not where high assurance is needed.

Theoretical cryptography explored hardware security from a formal perspective, with the goal of achieving an ideal leak-free hardware implementation of cryptographic primitives. The motivation for this work is today's extensive capabilities to learn hardware-protected secrets by observing the many side channels of hardware computation including power consumption, timing, electromagnetic radiation, and acoustic noise. Leakage-resilient cryptography was a very popular research direction in late-2000's, whose intensity since subsided. Works in this area typically assumed either a small secure component in hardware (small in chip area and computing only a minimal functionality, such as a pseudorandom function), or that leakage collected by an adversarial observer at any one time slice/epoch is *a priori* bounded. The goal was then to build provably-secure (i.e., leakage-free) hardware for computing a specific function based on these assumptions. A foundational paper by Micali and Reyzin (2004) introduced the notion of *physically observable cryptography* and proposed an influential theoretical framework to model side-channel attacks. In particular, they state and motivate the "only computation leaks information"

axiom used in much of leakage-resilient cryptography work. Juma and Vahlis (2010) show how to compute any function in a leak-free manner by using fully-homomorphic encryption and a single "leak-free" hardware token that samples from a distribution that does not depend on the protected key or the function that is evaluated on it. A corresponding line of research in the more practical hardware space acknowledges that real-world leakage functions are much stronger and more complex than what is often assumed by theoretical cryptography. In this line of work, heuristic, experimental and mitigation approaches are typical. Several MPC works have also taken advantage of (assumed) secure hardware tokens computing basic functions, such as PRF. Goals here include eliminating assumptions (e.g., Katz (2007) and Goyal *et al.* (2010)) or improving performance (Kolesnikov, 2010).