Knowledge-Oriented Secure Multiparty Computation

Piotr Mardziel, Michael Hicks,
Jonathan Katz
University of Maryland, College Park
{piotrm,mwh,jkatz}@cs.umd.edu

Mudhakar Srivatsa
IBM T.J. Watson Research Laboratory
msrivats@us.ibm.com

Abstract

Protocols for *secure multiparty computation* (SMC) allow a set of mutually distrusting parties to compute a function f of their private inputs while revealing nothing about their inputs beyond what is implied by the result. Depending on f, however, the result itself may reveal more information than parties are comfortable with. Almost all previous work on SMC treats f as given. Left unanswered is the question of how parties should decide whether it is "safe" for them to compute f in the first place.

We propose here a way to apply *belief tracking* to SMC in order to address exactly this question. In our approach, each participating party is able to reason about the increase in knowledge that other parties could gain as a result of computing f, and may choose not to participate (or participate only partially) so as to restrict that gain in knowledge. We develop two techniques—the *belief set* method and the *SMC belief tracking* method—prove them sound, and discuss their precision/performance tradeoffs using a series of experiments.

Categories and Subject Descriptors D.2.0 [*Software Engineering*]: Protection Mechanisms

General Terms Security, Language

1. Introduction

Consider a scenario where N parties $P_1, \ldots, P_N$ wish to compute some (known) function $f(s_1, \ldots, s_N)$ of their respective inputs, while ensuring *privacy* of their inputs to the extent possible. If these parties all trust some entity P_T, then each party P_i can simply send its input s_i to this trusted entity, who can in turn evaluate $f(s_1, \ldots, s_n)$ and return the result to each party. In the more general case, where f is a vector-valued function returning outputs $out_1, \ldots, out_N$, the trusted entity gives out_i to party P_i.

Cryptographic protocols for *secure multiparty computation* (SMC) [12, 7] allow the parties to accomplish the same task without the involvement of any trusted entity. (The reader can refer to a recent overview of SMC [10], or a textbook-level treatment [6].) That is, by running a distributed protocol amongst themselves the parties can learn the desired result $f(s_1, \ldots, s_N)$ (or, in the general case, each party P_i learns the result out_i) while ensuring that no information about other party's input is revealed beyond what is implied by the result(s). Section 2 provides further details about the precise notion of security that SMC protocols achieve.

Most work on SMC provides an answer to the question of *how* to compute f, but does not address the complementary question of *when* it is "safe" to compute f in the first place, i.e., when the output of f may reveal more information than parties are comfortable with. The two exceptions that we know of [4, 1] decide f's safety *independently* of the parties' inputs and in *isolation* of any (known or assumed) prior knowledge that parties have about each others' inputs.

However, the information implied by a query's result depends both on the parties' inputs and their prior knowledge. As an example of the former, suppose two parties want to compute the "less than or equal" function, $f(s_1, s_2) \overset{\text{def}}{=} s_1 \leq s_2$ with variables ranging in $\{1, \ldots, 10\}$. This function could reveal a lot about s_1 to P_2. If $s_2 = 1$ and $f(s_1, s_2)$ returns *true*, then P_2 learns that s_1 can only be the value 1. However, if $s_2 = 5$ then regardless of the output of the function, P_2 only learns that s_1 is one of 5 possibilities, a lower level of knowledge than in the first case.

On the other hand we may deem a pair of queries acceptable in isolation, but allowing their composition would be too revealing. For example, suppose the parties also want to compute "greater than or equal", $f_2(s_1, s_2) \overset{\text{def}}{=} s_1 \geq s_2$. When $s_2 = 5$, either query in isolation narrows the values of s_1 to a set of at least 4 possibilities from P_2's perspective. But if f_1 and f_2 both return *true*, P_2 can infer $s_1 = s_2 = 5$.

In recent work [11] we developed an approach to judging query safety called *knowledge-based security enforcement* (KBSE). In this paper we show how KBSE can be gen-

PLAS'12 June 15, Beijing, China.

eralized to SMC to address the limitations of current techniques listed above.

KBSE relies on reasoning about other parties' knowledge of one's own private data in order to determine whether a given function f is "safe" to compute in a given instance. Our previous work was in an *asymmetric* setting where only one of the parties (say, P_1) was concerned about privacy. The other parties' inputs could be revealed publicly, or at the very least be revealed to P_1; as such, the previous work did not involve SMC at all. At a high level, and specializing to the two-party case, party P_1 knows its own private data s_1 along with P_2's input s_2, and also maintains a *belief* about P_2's knowledge of s_1 (represented as a probability distribution δ). Before agreeing to compute a function $f(s_1, s_2)$, P_1 determines whether computing the residual function $f(\cdot, s_2)$ would reveal "too much information" as determined according to a threshold $0 < t_1 \leq 1$ set by P_1. In particular, P_1 will not compute the function if P_2's belief about the likelihood of a possible secret value (including the actual secret s_1) increases above t_1.[1] If P_1 does reveal $f(s_1, s_2)$ then it determines what P_2 will learn from the output and revises its estimate δ_2 of P_2's knowledge accordingly. It will use this new estimate when considering subsequent functions. (KBSE is reviewed in Section 3.)

In our prior work, P_1's determination as to whether it should agree to compute f relied in an essential way on the fact that P_1 knows the input s_2 of the other party. In the SMC setting the privacy of *all* parties' inputs should be preserved, so our prior techniques cannot be applied directly. In this paper we initiate the idea of combining KBSE and SMC, in order to address the question of when it is safe to compute some function f of multiple parties' inputs.

We present two techniques (Section 4). The first, which we call the *belief set* method, works as follows. Each P_i maintains an estimate of the *set* of distributions Δ_j for each other principal P_j, one for each possible valuation of s_j (assigned probability 1). In short, P_j's actual belief δ_j is a member of the set Δ_j. The same basic procedure as in the prior work, lifted from distributions to sets of distributions, can then be applied by each P_i, and if all agree to participate, they perform the function evaluation via SMC.

The second technique we call *SMC belief tracking*. Rather than have each principal P_i perform the KBSE procedure individually before the SMC takes place, the KBSE procedure is performed *within the SMC itself*. If the SMC-KBSE procedure determines that any of the thresholds t_i will be exceeded by sending a response to P_j then P_j receives a rejection, rather than the actual answer. However, because P_k's knowledge will be different, it could receive a proper answer. By performing KBSE within the SMC, we can look at the *actual* secret values of each of the participants and by accepting/rejecting selectively, we can ensure that no information is revealed by rejection. As we show in Section 5 using a series of experiments with our proof-of-concept implementation, SMC belief tracking is strictly more precise (in that fewer queries will be rejected) than belief sets. On the other hand, SMC is known to be very slow, and so implementing KBSE as an SMC could be quite costly. We leave exploration of implementation strategies to future work.

In summary, the main contribution of this paper is a pair of techniques for evaluating the safety of SMC computations. To our knowledge, ours is the first work to consider the question of safety in the context parties' actual secrets and prior knowledge, and approach that should allow more queries to be answered safely, even in composition.

2. Secure Multiparty Computation

This section presents basic background on secure multiparty computation (a completely formal treatment of the security provided by SMC is beyond the scope of this paper). Throughout this paper we assume that all parties are *semi-honest*. This means that they run any specified protocol exactly as prescribed, but may try to infer information about other parties' inputs based on their *view* of the protocol execution. (A party's view consists of its local state, along with all messages that it sent or received.) We also assume that parties do not collude. SMC can be extended to malicious parties who behave arbitrarily, as well as to handle collusion, but these complicate the treatment and are tangential to our main thrust.

As described in the introduction, we consider a scenario where N mutually distrusting parties $P_1, \ldots, P_N$ wish to compute some (known) function $f(s_1, \ldots, s_N)$ of their respective inputs, while ensuring *privacy* of their inputs to the extent possible. In an ideal scenario, the parties would all have access to a trusted entity P_T who would compute the function on their behalf. That is, each party P_i would simply send its input s_i to P_T, who would in turn evaluate $(out_1, \ldots, out_n) = f(s_1, \ldots, s_n)$ and return the result out_i to party P_i. We write $out = f(...)$ if the same output is sent to all participants. If f is a probabilistic function, then P_T evaluates it using uniform random choices.

Fix some distributed protocol Π that computes f. (This just means that when the parties run the protocol using their inputs $s_1, \ldots, s_N$, the protocol terminates with each party holding output out_i.) We say that Π is *secure* if it emulates the ideal computation of f described above (where a trusted entity is available). Specifically, an execution of Π should reveal no information beyond what is revealed in the ideal computation.[2] This is formally defined by requiring that any

[1] The release criterion considers all possible values for s_1 — and not just the actual value of s_1 — so that a refusal to participate does not leak any information about s_1.

[2] Readers who are familiar with SMC may note that this definition is slightly simpler than usual. The reason is that we are considering semi-honest security, and in this paragraph assume a deterministic function for simplicity. We are also glossing over various technical subtleties that are inessential to get the main point across.

party in the ideal world can sample from a distribution that is "equivalent" to the distribution of that party's view in a real-world execution of Π. Since any party P_i in the ideal world knows only its own input s_i and the output out_i that it received from P_T, this implies that Π achieves the level of privacy desired. We stress that not only is no information (beyond the output) about any *single* party's input is revealed, but also no *joint* information about several parties' inputs is revealed either (just as in the ideal world).

The cryptographic literature considers several notions of what it means for two distributions D, D' to be "equivalent". The simplest notion is to require D, D' to be *identical*. If this is the case for the distributions described above, then Π is said to achieve *perfect* security. Alternately, we may require that D, D' be indistinguishable by computationally bounded algorithms. (We omit a formal definition, though remark that this notion of indistinguishability is pervasive in all of cryptography, beyond SMC.) In this case, we say that Π achieves *computational* security. Perfect security is achievable for $N \geq 3$, whereas only computational security is possible for $N = 2$.

In the remainder of the paper we assume that the secrets s_i remains fixed during a sequence of computations, so that information gained about s_i from one computation carries over to the next. We also assume that the P_i have no means to communicate outside the SMC, so that what can be learned about a particular secret depends only on the functions computed via an SMC. We leave relaxation of these restrictions to future work.

3. Knowledge-Based Security Policies

Our goal is to devise a method whereby each principal can determine whether participation in an SMC would reveal too much information about its secret. In prior work [11] we developed a solution for a special case of this problem. In this case we have two principals, P_1 and P_2, and only P_1 has a secret value x_1. In this situation, P_2 wishes to compute some function Q of x_1, and P_1 only wishes to proceed if P_2 remains uncertain about x_1 upon learning the result. If this is the case, P_1 computes the result n and sends it back to P_2. If not, it sends a rejection message.

The key question is: how does P_1 reason what P_2 might learn about x_1 based on the output of Q? To answer this question, we adopted the approach of Clarkson et al. [2]. In their approach, P_2 has a *belief* about the possible values of x_1. They show how that belief can be revised upon learning the output of a function over that secret. In our approach, P_1 *estimates* what P_2 might know about x_1 (e.g., that it is uniformly distributed), and then uses Clarkson et al.'s method to determine how much information P_2 might gain from the answer to Q. If this information exceeds a threshold, the query is rejected.

In the remainder of this section, we describe Clarkson et al's technique, and then our application of it to knowledge-based security enforcement. In the next section, we show how this approach can be generalized to the SMC setting.

$$
\begin{array}{llll}
\textit{Variables} & x & \in & \mathbf{Var} \\
\textit{Integers} & n, s, o & \in & \mathbb{Z} \\
\textit{Rationals} & r & \in & \mathbb{Q} \\
\textit{Arith.ops} & aop & ::= & + \mid \times \mid - \\
\textit{Rel.ops} & relop & ::= & \leq \mid < \mid = \mid \neq \mid \cdots \\
\textit{Arith.exps} & E & ::= & x \mid n \mid E_1 \; aop \; E_2 \\
\textit{Bool.exps} & B & ::= & E_1 \; relop \; E_2 \mid \\
 & & & B_1 \wedge B_2 \mid B_1 \vee B_2 \mid \neg B \\
\textit{Statements} & Q, S & ::= & \mathsf{skip} \mid x \ := \ E \mid \\
 & & & \mathsf{if}\ B\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2 \mid \\
 & & & \mathsf{pif}\ r\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2 \mid \\
 & & & S_1 \ ; \ S_2 \mid \mathsf{while}\ B\ \mathsf{do}\ S
\end{array}
$$

Figure 1. Core language syntax

$$
\begin{array}{rcl}
[\![\mathsf{skip}]\!]\delta & = & \delta \\
[\![x \ := \ E]\!]\delta & = & \delta\,[x \rightarrow E] \\
[\![\mathsf{if}\ B\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!]\delta & = & [\![S_1]\!](\delta|B) + [\![S_2]\!](\delta|\neg B) \\
[\![\mathsf{pif}\ q\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!]\delta & = & [\![S_1]\!](q \cdot \delta) + [\![S_2]\!]((1-q) \cdot \delta) \\
[\![S_1 \ ; \ S_2]\!]\delta & = & [\![S_2]\!]([\![S_1]\!]\delta) \\
[\![\mathsf{while}\ B\ \mathsf{do}\ S]\!] & = & \mathrm{lfp}\,[\lambda f : \ \mathbf{Dist} \rightarrow \mathbf{Dist}.\ \lambda\delta. \\
 & & \quad f\,([\![S]\!](\delta|B)) + (\delta|\neg B)]
\end{array}
$$

where

$$
\begin{array}{ll}
\delta\,[x \rightarrow E] & \overset{\text{def}}{=} \lambda\sigma.\ \sum_{\tau \ | \ \tau[x \rightarrow [\![E]\!]\tau] = \sigma} \delta(\tau) \\
\delta_1 + \delta_2 & \overset{\text{def}}{=} \lambda\sigma.\ \delta_1(\sigma) + \delta_2(\sigma) \\
\delta|B & \overset{\text{def}}{=} \lambda\sigma.\ \mathbf{if}\ [\![B]\!]\sigma\ \mathbf{then}\ \delta(\sigma)\ \mathbf{else}\ 0 \\
p \cdot \delta & \overset{\text{def}}{=} \lambda\sigma.\ p \cdot \delta(\sigma) \\
\|\delta\| & \overset{\text{def}}{=} \sum_\sigma \delta(\sigma) \\
\textit{normal}(\delta) & \overset{\text{def}}{=} \frac{1}{\|\delta\|} \cdot \delta \\
\delta \,\|\, B & \overset{\text{def}}{=} \textit{normal}(\delta|B)
\end{array}
$$

Figure 2. Probabilistic semantics for the core language

3.1 Clarkson et al.'s knowledge estimation

The programming language we use for computations is given in Figure 1. A computation is defined by a statement S whose standard semantics can be viewed as a relation between states: we write $[\![S]\!]\sigma = \sigma'$ to mean that running statement S with input state σ produces output state σ', where states map variables to integers:

$$\sigma, \tau \in \mathbf{State} \overset{\text{def}}{=} \mathbf{Var} \rightarrow \mathbb{Z}$$

Sometimes we consider states with domains restricted to a subset of variables V, in which case we write $\sigma_V \in \mathbf{State}_V \overset{\text{def}}{=} V \rightarrow \mathbb{Z}$. We will write $\{x_1 = s_1, ..., x_n = s_n\}$ to represent a state σ whose domain is $\{x_1, ..., x_n\}$ such that $\sigma(x_1) = s_1$, $\sigma(x_2) = s_2$, etc. We may also *project* states to a set of variables V:

$$\sigma \upharpoonright V \overset{\text{def}}{=} \lambda x \in \mathbf{Var}_V.\ \sigma(x)$$

The language is essentially standard. The semantics of the statement form pif r then S_1 else S_2 is non-deterministic: the result is that of S_1 with probability r, and S_2 with probability $1 - r$.

In our setting, we limit our attention to *queries* in this language. A query as a statement Q that can read, but not write, free variables $x_1, ..., x_n$ (i.e., these are set in the initial state σ), and sets the output to the variable out.

Example 1. As an example, consider the following query:

$$\begin{array}{rcl} Q_0 & \overset{\text{def}}{=} & \mathsf{if}\ x_1 \geq 7 \\ & & \quad \mathsf{then}\ out\ := \ \mathsf{True} \\ & & \quad \mathsf{else}\ out\ := \ \mathsf{False} \end{array}$$

Given an input state $\sigma = \{x_1 = 3\}$, we have that $[\![Q_0]\!]\sigma = \sigma'$ where $\sigma' = \{x_1 = 3, out = \mathsf{False}\}$.

A belief is represented as a probability distribution, which is conceptually a map from states to positive real numbers representing probabilities (in range $[0, 1]$).

$$\delta \in \mathbf{Dist} \overset{\text{def}}{=} \mathbf{State} \rightarrow \mathbb{R}+$$

In what follows, we often notate distributions using lambda terms; e.g., we write $\lambda\sigma.\mathbf{if}\ \sigma(x_1) = 3\ \mathbf{then}\ 1\ \mathbf{else}\ 0$ to represent the point distribution assigning probability 1 to the state σ in which x_1 is 3, and probability 0 to all other states.

Given a principal's initial belief, Clarkson et al. define a mechanism for *revising* that belief according to the output of a query. This works as follows. First, a principal evaluates the query according to its belief using the *probabilistic semantics* given in Figure 2. This semantics is standard (cf. Clarkson et al. [2]) so, due to space constraints, we do not describe it in detail here. It suffices to understand that $[\![S]\!]\delta$ represents probabilistic execution: we write $[\![S]\!]\delta = \delta'$ to say that the distribution over program states after executing S with δ is δ'. We may view δ' as a prediction of the likelihood of the possible input states according to the possible output states. Upon seeing the actual output of the query, the principal can *revise* this prediction; we write such revision as $[\![S]\!]\delta \,||\, (out = n)$, where $out = n$ is a boolean expression B and n is the actual observed output. The definition of revision $\delta \,||\, B$ is given at the bottom of Figure 2. The revised belief can be used as the prior belief for a future query. The revision operation itself is a conditioning, which usually results in a distribution with a mass not equal to 1, followed by a normalization, which produces a real distribution.

Returning to Example 1, suppose that x_1 represents P_1's secret value, and P_2's belief δ_2 is as follows

$$\delta_2 \overset{\text{def}}{=} \lambda\sigma.\ \mathbf{if}\ \sigma(x_1) < 0\ \mathbf{or}\ \sigma(x_2) \geq 10\ \mathbf{then}\ 0\ \mathbf{else}\ 1/10$$

Thus, δ_2 is a function from states to real numbers implementing a uniform distribution: if x_1's value in σ is between 0 and 9 then σ is given probability $1/10$, otherwise it is given probability 0. To revise δ_2 according to the actual output $out = \mathsf{False}$, principal P_2 first computes $[\![q_0]\!]\delta_2 = \delta_2'$, which when simplified can be written

$$\begin{array}{l} \delta_2' \overset{\text{def}}{=} \lambda\sigma.\mathbf{if}\ \sigma(x_1) < 0\ \mathbf{or}\ \sigma(x_2) \geq 10\ \mathbf{then}\ 0 \\ \qquad\quad \mathbf{else\ if}\ \sigma(out) = \mathsf{True}\ \mathbf{and}\ \sigma(x_1) \geq 7\ \mathbf{then}\ 1/10 \\ \qquad\quad \mathbf{else\ if}\ \sigma(out) = \mathsf{False}\ \mathbf{and}\ \sigma(x_1) < 7\ \mathbf{then}\ 1/10 \\ \qquad\quad \mathbf{else}\ 0 \end{array}$$

Revising δ_2' under the assumption that $out = \mathsf{False}$ would produce the following (simplified) distribution:

$$\begin{array}{l} \delta_2' \,||\, (out = \mathsf{False}) \overset{\text{def}}{=} \\ \qquad \lambda\sigma.\ \mathbf{if}\ \sigma(x_1) < 7\ \mathbf{or}\ \sigma(x_2) \geq 10\ \mathbf{then}\ 0\ \mathbf{else}\ 1/7 \end{array}$$

Soundness. Clarkson et al. show that the probabilistic semantics and revision exactly model the changing belief of an adversary as it learns outputs of the queries, assuming no other channel of information flow exists, and the adversary is rational and has unbounded computational power.

Theorem 2 (Theorem 1 of [2])**.** *A rational, computationally unbounded agent, having belief δ about x_1, updates its belief to δ' after learning output n of a query Q, with no other channels, where δ' is $[\![Q]\!]\delta \,||\, (out = n)$.*

3.2 Enforcing knowledge-based security policies

Our prior work [11] uses Clarkson et al's technique as a key building block for handling the scenario given in Figure 3. Here, in step 1 P_2 sends a query Q and a state σ to P_1. In step 2, P_1 decides whether Q is safe to compute, and if so, executes $[\![Q]\!]\sigma' = \sigma''$, where σ' is σ with the added mapping of x_1 to P_1's secret s_1. In step 3, P_1 sends back the result

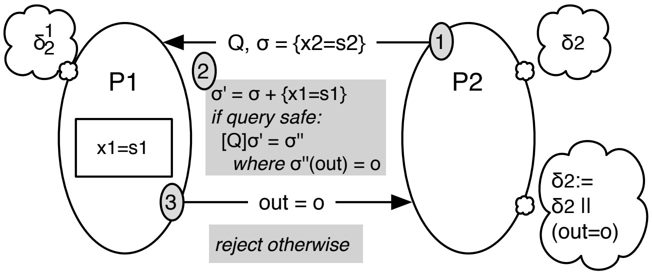


Figure 3. Asymmetric belief tracking

```
tcheck($q, \delta_i, t_j, x_j$) $\overset{\text{def}}{=}$
$\underline{\delta_i} := [\![q]\!]\delta_i$
forall possible outputs $o$
    $\hat{\delta}_i := (\underline{\delta_i} \,||\, (out = o)) \upharpoonright \{x_j\}$
    if $\exists n.\ \hat{\delta}_i(\{x_j = n\}) > t_j$ then
        return reject
return accept
```

Figure 4. Threshold policy decision, tcheck

$o = \sigma''(out)$ if the query was safe, and otherwise rejects the query. P_2 revises its belief δ_2 based on the outcome.

The main question to answer is how P_1 determines whether Q is safe, i.e., whether it "reveals too much information." We propose that principal P_1 assign to its secret a *knowledge threshold* t_1, where $0 < t_1 \leq 1$, interpreted to mean that P_2 should never be certain of P_1's secret with probability greater than t_1. Returning to Example 1, suppose that P_1's knowledge threshold $t_1 = 1/10$ and $x_1 = 3$. Running Q_0 produces False, and P_2's revised belief δ_2' assigns to the state $\{x_1 = 3, out = \mathsf{False}\}$ the probability $1/7$, which exceeds the threshold. As such P_1 ought to reject the query. On the other hand, if the threshold was $1/2$, then the query could be accepted.

Keeping this intuition in mind, here is how the part notated *is the query safe* in Figure 3 is implemented. First, P_1 *estimates* P_2's belief δ_2 about P_1's secret value. We write δ_2^1 to indicate this estimate.[3] Then P_1 calls $\mathsf{tcheck}(Q, \delta_2^1, t_1, x_1)$, the pseudocode for which is given in Figure 4. Here, δ_i is bound to P_1's estimate δ_2^1, while t_j and x_j are bound to t_i and x_i (that is, the *variable name* x_i, not the value it is bound to), respectively.

On line 1, P_1 probabilistically executes $[\![Q]\!]\delta_i$ producing $\underline{\delta_i}$. Then, for each possible output o (line 2), P_1 can revise the belief, $\underline{\delta_i} \,\|\, (out = o)$, from which we can project states to involve only secret x_1, written $\hat{\delta_i} = (\underline{\delta_i}|(out = o)) \upharpoonright \{x_1\}$ (line 3). We explain shortly why every possible output must be considered, rather than just the output for P_1's actual secret value. On line 4, we check whether for o and corresponding revised belief $\hat{\hat{\delta}}_i$ there exists a possible value n such that $(\hat{\hat{\delta}}_i)(\{x_1 = n\}) > t_1$. If so, the query Q must be rejected, to avoid leaking too much information (line 5). Otherwise, the query is acceptable (line 6).

If $\mathsf{tcheck}(Q, \delta_2^1, t_1, x_1)$ returns *accept* then P_1 can execute the query, send back the result, and update its estimate δ_2^1 to be $\underline{\delta_2^1} \,\|\, (out = o)$.

Avoiding leakage due to query rejection. Line 2 in Figure 4 requires we consider all possible outputs o. At first glance, doing so seems unnecessarily conservative. For Example 1, suppose that $t_1 = 1/5$ and $\delta_1^2 = \delta_2$; then executing $\mathsf{tcheck}(Q_0, \delta_2^1, t_1, x_1)$ would produce *reject*. But if the actual secret is $x_1 = 3$, then we have already established that answering the query (with False) results in δ_2 being revised to assign $\{x = 3, out = \mathsf{False}\}$ probability $1/7$ which is below the threshold. On the other hand, suppose that x_1 was 8 instead of 3, in which case answering the query with True would cause P_2's revised belief to ascribe probability $1/3$ to $\{x_1 = 8, out = \mathsf{True}\}$, which exceeds the threshold $t_1 = 1/5$. But if P_1 rejects the query, and P_2 knows threshold t_1 it will be able to infer that the only reason for rejection would be that the answer would have been True. Even if t_1 is not known directly, it can be inferred by enough queries to eventually make this sort of determination. P_1 avoids this situation by rejecting any query for which there exists a secret that could be compromised by the answer, even if that does not happen to be its secret. This approach results in P_1 deciding to allow a query or not independetly of his true secret value. Such policy decisions are *simulatable* [9] in that P_2 could have determined on their own whether P_1 will reject the query, hence learning of P_1's decision tells them nothing.

4. Enforcing knowledge thresholds for SMC

In this section we show how to generalize knowledge-based enforcement from the single-secret scenario given in Figure 3 to the multi-secret setting of SMC. In this setting, there are N principals, $P_1, ..., P_N$ each with a secret $x_1 = s_1, ..., x_N = s_N$. Each P_i maintains a belief δ_i about the possible values of the other participating principals' secrets. In addition, each P_i has a knowledge threshold t_i that bounds the certainty that the other principals can have about its secret's value.

Next we present an example to illustrate how belief estimation is adapted to the SMC case, and then we use this example to illustrate two possible methods we have devised for enforcing the knowledge threshold, the *belief set* method (Section 4.2) and the *SMC belief tracking* method (Section 4.4). We prove both methods are sound and discuss their tradeoffs in Section 5.

4.1 Running example

Suppose we have three principals, P_1, P_2, and P_3, each with a net worth $x_1 = 20$, $x_2 = 15$, and $x_3 = 17$, in millions of dollars, respectively. Suppose they wish to compute Q_1 which determines whether P_1 is the richest:

$$
\begin{aligned}
Q_1 \;\overset{\text{def}}{=}\;\; & \mathsf{if}\ x_1 \geq x_2 \wedge x_1 \geq x_3 \\
& \mathsf{then}\ out\ :=\ \mathsf{True} \\
& \mathsf{else}\ out\ :=\ \mathsf{False}
\end{aligned}
$$

Using the idealized view, each of P_1, P_2, and P_3 can be seen as sending their secrets to P_T, which initializes σ such that $\sigma(x_1) = 20$, $\sigma(x_2) = 15$, and $\sigma(x_3) = 17$. Running Q_1 using σ produces an output state σ' such that $\sigma'(out) = \mathsf{True}$.

Now suppose that P_1 believes that both P_2 and P_3 have at least \$10 million, but less than \$100 million, with each case equally likely. Thus principal P_1's belief is defined as

$$
\begin{aligned}
\delta_1 \overset{\text{def}}{=} \lambda\sigma.\mathbf{if}\ & \sigma(x_2) < 10\ \mathbf{or}\ \sigma(x_2) > 100\ \mathbf{or} \\
& \sigma(x_3) < 10\ \mathbf{or}\ \sigma(x_3) > 100\ \mathbf{or} \\
& \sigma(x_1) \neq 20\ \mathbf{then}\ 0\ \mathbf{else}\ 1/8281
\end{aligned}
$$

States which ascribe either x_2 or x_3 a net worth outside the expected range, or ascribe x_1 to the wrong value, are considered impossible, and every one of the remaining 8281

[3] How P_1 comes by this estimate is beyond the scope of this paper, but we point out that for many kinds of data, good estimates are easy to come by. For example, generic distributions over personal information like gender, birthday, social security number, income, etc. can be gained from census data or other public and private repositories (e.g., Facebook demographics).

(that is 91×91) states is given probability $1/8281$. The beliefs of P_2 and P_3 are defined similarly.

Belief revision proceeds as before: once P_T performs the computation and sends the result, each P_i revises its belief. For our example query Q_1, principal P_1 would perform $[\![Q_1]\!]\delta_1 = \delta_1''$ and since the output of the query is True, then revision produces $\delta_1' = [\![Q_1]\!]\delta_1 | (out = \mathsf{True})$. This revised belief additionally disregards states that ascribe x_2 or x_3 to values greater than P_1's own wealth, which is \$20M:

$$\begin{aligned}\delta_1' \overset{\text{def}}{=} \lambda\sigma.\textbf{if } & \sigma(x_2) < 10 \textbf{ or } \sigma(x_2) > 20 \textbf{ or} \\ & \sigma(x_3) < 10 \textbf{ or } \sigma(x_3) > 20 \textbf{ or} \\ & \sigma(x_1) \neq 20 \textbf{ then } 0 \textbf{ else } 1/121\end{aligned}$$

The revised beliefs of P_2 and P_3 will be less specific, since each will simply know that P_1's wealth is at least their own and no less than the rest of the parties.

4.2 Knowledge-based security with belief sets

Now we wish to generalize threshold enforcement, as described in Section 3.2, to SMC. In the simpler setting P_1 maintained an estimate δ_2^1 of P_2's belief δ_2. In the SMC setting we might imagine that each P_j maintains a belief estimate δ_i^j and then performs $\mathsf{tcheck}(q, \delta_i^j, t_j, x_j)$ for all $i \neq j$. If each of these checks succeeds, then P_j is willing to participate.

The snag is that P_j cannot accurately initialize δ_i^j for all $i \neq j$ because it cannot directly represent what P_i knows about x_i—that is, its exact value. So the question is: how can P_j estimate the potential gain in P_i's knowledge about x_j after running query without knowing x_i?

One approach to solving this problem, which we call the *belief set* method, is the following. P_j follows roughly the same procedure as above, but instead of maintaining a single distribution δ_i^j for each remote party P_i, it maintains a *set* of distributions where each distribution in the set applies to a particular valuation of x_i. As a first cut, suppose that P_j initializes this set to be as follows:

$$\Delta_i^j \overset{\text{def}}{=} \{\delta_i^j \,\|\, (x_i = v) \;:\; v = \sigma(x_i), \sigma \in \textit{support}(\delta_j \restriction \{x_i\})\}$$

Thus Δ_i^j is a set of possible distributions, one per possible valuation of x_i that P_j thinks is possible according to its belief δ_j.

However, this method for initializing the set is not quite expressive enough, since it may fail to take into account correlations among beliefs of multiple principals. For example, if it were known (by all principals) that only one of the principals in the running example can have secret value equal to 15, then P_2 would know initially, based on this own secret $x_2 = 15$, that P_1's value x_1 cannot be 15. However, P_1 cannot arrive at this conclusion without knowing x_2, which is, of course, outside of its knowledge initially.

Therefore, we define the initial belief set using a distribution δ over *all* principals' secret data which sufficiently captures any correlations in those secrets. Such a distribution can then be used, given some valuations of secret variables, to derive what a principal's initial belief would be.

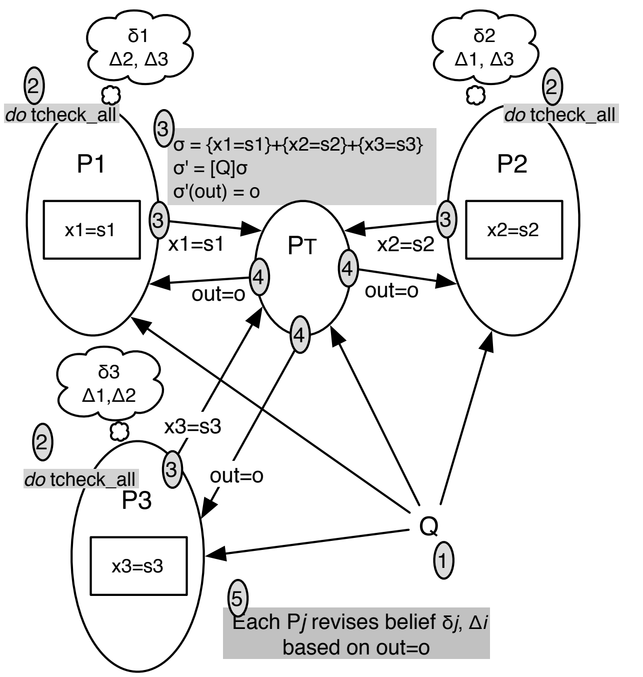


Figure 5. Threshold enforcement for SMC using belief sets

```
tcheck_all(q, j) $\overset{\text{def}}{=}$
1  forall $i \in 1..n$ with $i \neq j$
2     tcheck($q, \Delta_i, t_j, x_j$)
3  if all threshold checks succeed then
4     agree to participate
5  else
6     refuse to participate
```

Figure 6. tcheck_all check for belief set enforcement

$$\Delta_i \overset{\text{def}}{=} \{\delta \,\|\, (x_i = v) \;:\; v = \sigma(x_i), \sigma \in \textit{support}(\delta \restriction \{x_i\})\}$$

Since we are starting from a globally held belief δ, there is no need to distinguish Δ_i^j from Δ_i^k—they are the same Δ_i.

Now each P_j follows the procedure depicted in Figure 5 for the idealized view (with a trusted principal P_T). First, the principals agree on the query Q. Second, each principal P_j performs the threshold check $\mathsf{tcheck\_all}(Q, j)$, whose code is given in Figure 6. Notice that calls to tcheck(...) on line 2 are with the set Δ_i, rather than a single distribution δ_i^j. The definitions of the operations in the pseudocode in Figure 4, when applied to sets Δ rather single elements δ, are defined in Figure 7. In all but the last case, these operations are just straightforward liftings of the operations on single distributions. For $\Delta(\sigma)$, we return the highest probability for σ of those ascribed to it by distributions in Δ, to assure that our decision to participate or not is safe. Also note that we will always be dealing with non-empty Δ, hence the

Semantics

$$[\![S]\!]\Delta = \{[\![S]\!]\delta \ : \ \delta \in \Delta\}$$

Operations

$$\begin{array}{ll} \Delta \upharpoonright V & \overset{\text{def}}{=} \{(\delta \upharpoonright V) \ : \ \delta \in \Delta\} \\ \mathit{normal}(\Delta) & \overset{\text{def}}{=} \{\mathit{normal}(\delta) \ : \ \delta \in \Delta \ , \ \|\delta\| > 0\} \\ \Delta \,\|\, B & \overset{\text{def}}{=} \mathit{normal}(\{(\delta | B) \ : \ \delta \in \Delta\}) \\ \Delta(\sigma) & \overset{\text{def}}{=} \max_{\delta \in \Delta} \delta(\sigma) \end{array}$$

Figure 7. Probabilistic semantics using sets of distributions

maximum probability is sufficiently defined. On the other hand, the normalization procedure for distributions δ is only well defined whenever $\|\delta\| > 0$. Because of this, we make sure the normalization for distribution sets only normalizes the normalizable distributions, and discards the rest. The way in which some member distributions of Δ could become non-normalizable, that is, having mass of 0, is by way of the conditioning operation, where the condition is inconsistent with all possible states in the distribution.

In the third step, if the query is acceptable for all P_j, each sends its secret $x_j = s_j$ to P_T, which executes Q using the secret state σ constructed from each secret. Fourth, the result o is sent back to each principal. Finally, as usual, P_1 revises each of its estimates Δ_i and its own belief δ_j. Note that all principals make the same update for Δ_i, hence there really is only one Δ_i, known by all, estimating P_i's knowledge.

While we have depicted this procedure in the idealized view of SMC, it is easy to see that we can simply implement steps 3 and 4 as a normal SMC and the remainder of the procedure is unchanged.

4.3 Soundness of belief sets

Now we can show that the belief set procedure is *sound*, in that for all P_i, participating or not participating in a query will never increase another P_j's certainty about P_i's secret above its threshold t_i.

Remark 3. Suppose principals $P_1, ..., P_N$ wish to execute a query Q. The secret state $\sigma_s = \{x_1 = s_1, ..., x_N = s_N\}$ contains all their secrets. Assume that for each P_i:

1. P_i has a belief δ_i.
2. P_i's belief δ_i is consistent with σ_s, that is, $\delta_i(\sigma_s) > 0$.
3. P_i's belief δ_i is within the public estimate of his knowledge, that is, $\delta_i \in \Delta_i$.

Suppose $[\![Q]\!]\sigma_s = \sigma'_s$ such that $\sigma'_s(out) = o$. That is, the actual output of the query Q is o. Then, the belief of each agent, after learning the output, is $\delta_i \,\|\, (out = o)$, and is a member of the estimated set $\Delta_i \,\|\, (out = o)$.

Proof. Theorem 2 tells us that $\delta'_i = [\![Q]\!]\delta_i \,\|\, (out = o)$ are the new beliefs of the principals, having learned that $out = o$. By assumption we had $\delta_i \in \Delta_i$, and since δ_i was consistent with σ_s, it must be that δ'_i is consistent (having non-zero mass), and therefore $\delta'_i \in \Delta'_i = \Delta_i \,\|\, (out = o)$. □

This remark is merely a lifting of Theorem 2 to sets of beliefs. The more interesting point arises when the principals are also interested in enforcing a knowledge threshold.

Lemma 4. *Suppose the same premise as Remark 3. Also suppose that policy thresholds t_i are public, and $[\![Q]\!]\sigma_s = \sigma'_s$ such that $\sigma'_s(out) = o$. That is, the actual output of the query Q is o, and each P_i learns either*

- *the output o of the query, or*
- *which principals P_j rejected the query.*

Then, the belief of each agent, in the first case is $\delta_i \,\|\, (out = o)$, and is within the estimate $\Delta_i \,\|\, (out = o)$, or in the second case, remains at δ_i.

Proof. The lemma effectively states that the policy decisions have no effect on the beliefs; if a query is rejected, learning which principals rejected it reveals nothing. Similarly, if the query is not rejected, the additional information each principal gets (that no one rejected the query), also does not change the belief.

The lemma holds due to the simple fact that the policy decisions do not depend on private information (see Figure 6), every single principal could determine, on their own, whether another principal would reject a query. Thus the policy decisions, as a whole, are a simulatable procedure. The rest follows from Remark 3. □

Some subtleties are worth mentioning. First, a premise of the lemma is that Δ_i are known by all principals. This fact needs to remain as the query is answered so the same premise will hold for the next query. Fortunately this is the case, as the revised belief sets in the case of policy success, $\Delta_i \,\|\, out = o$ are also known by all participants, as o is known, and so are the initial Δ_i.

A second subtlety is that the queries themselves must be chosen independent of anyone's secret. In some situations, where the principals are actively attempting to maximize their knowledge, and are allowed to propose queries to accomplish this, the query choice can be revealing. This problem is beyond the scope of this work, and we will merely assume the query choice is independent of secrets.

4.4 SMC belief tracking: ideal world

Now we present an alternative to the belief set method, in which the decision to participate or not, involving checking thresholds after belief revision, takes place *within the SMC itself*. As such, we call this method *SMC belief tracking*. Once again we present the algorithm using the ideal world with a trusted third party P_T. The steps are shown in Figure 8. The first step is that each P_i presents its secret $x_i = s_i$ to P_T, along with the collective belief δ. Principal P_T then initializes the computation state by calling

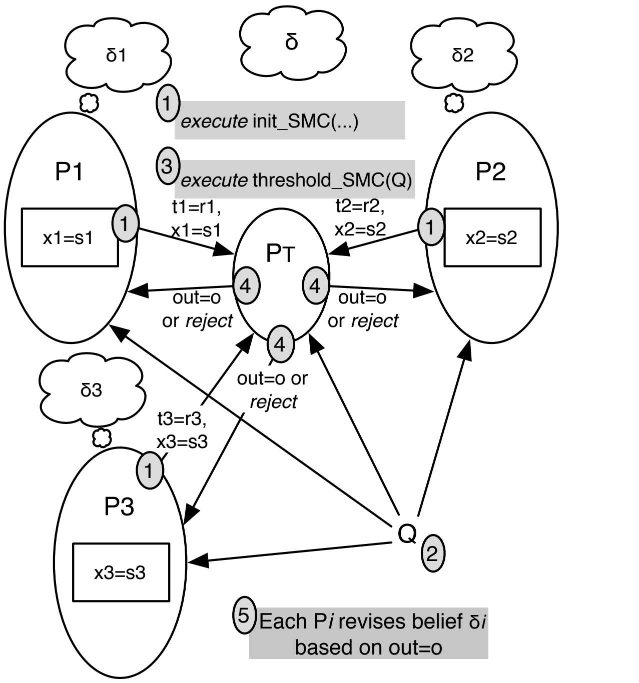


Figure 8. SMC belief tracking scenario (ideal view)

```
init_SMC($s_1 \ldots s_N, r_1 \ldots r_N, \delta$) $\stackrel{\text{def}}{=}$
1    $\sigma_s := \{x_1 = s_1, \ldots, x_N = s_N\}$
2    $\delta_1 := \delta \,\|\, (x_1 = s_1);\ t_1 := r_1$
       ...
n+1  $\delta_N := \delta \,\|\, (x_N = s_N);\ t_N := r_N$

threshold_SMC($Q$) $\stackrel{\text{def}}{=}$
1    $o := ([\![Q]\!]\sigma_s)(out)$
2    forall $j \in 1..n$
3      forall $i \in 1..n$ with $i \neq j$
4        tcheck($Q, \delta_j, t_i, x_i$)
5      if all threshold checks succeed then
6        $\delta_j := [\![Q]\!]\delta_j \,\|\, (out = o)$
7        return $o$ to $P_j$
8      else
9        return reject to $P_j$
```

Figure 9. SMC belief tracking (ideal view)

init_SMC($s_1 \ldots s_N, \delta$), given in Figure 9. On line 1, this code initializes the secret state σ_s that contains all of the secrets. On lines 2..(n+1), it initializes each principal P_i's belief as in the belief set case, by specializing δ with the knowledge unique to P_i. It also initializes each threshold t_i to r_i.

In step 2 (of the diagram), the query Q is made available to P_T, which then runs (in step 3) threshold_SMC(Q), also shown in Figure 9. On line 1 we compute the actual output o for the query, based on the secret state. On line 2 we loop over each principal P_j. The remainder of the code aims to decide whether answering the query and sending the result to P_j would reveal too much information; if not, we send P_j the answer o (line 7) and otherwise we reject.

Returning to the body of the loop, the next step is to make sure that for every P_i (line 3) its threshold check (Figure 4) will not reject P_j. That is, given the query q and the estimated knowledge of P_j, we make sure that the answer to the query will not reveal too much about P_i's secret x_i (where by "too much" we mean P_j's certainty about P_i's possible secret exceeds threshold t_i). Assuming all P_i threshold checks succeed (line 5), we then revise the P_j's belief according to the output o (line 6), which we then send to P_j (step 4 in the diagram). No revision is done on P_j's belief if the query is rejected for P_j. Finally, each principal revises its own belief δ_j based on the output.

We can repeat steps 2–5 for each subsequent query Q', and P_T will use any beliefs δ_j revised from the run of Q. By performing threshold_SMC as part of an SMC, no participant P_i is ever shown the opposite's secret, and yet an accurate determination is made for each about whether to participate.

Importantly, the fact that P_j receives a proper answer or *reject* is not (directly) observed by any other P_j; such an observation could reveal information to P_j about x_i. For example, suppose $Q_2 \stackrel{\text{def}}{=} x_1 \leq x_2$ and both secrets are (believed to be) between 0 and 9. If $x_2 = 0$ then $[\![Q_2]\!]\sigma_s$ will return True only when x_1 is also 0. Supposing $t_1 = 3/5$, then P_2 should receive *reject* since there exists a valuation of x_1 (that is, 0) such that P_2 could guess x_1 with probability greater than 3/5. Similar reasoning would argue for reject if $x_2 = 9$, but acceptance in all other cases. As such, if P_1 observes that P_2 receives *reject*, it knows that x_2 must be either 0 or 9, independent of t_2; as such, if $t_2 < 1/2$ we have violated the threshold by revealing the result of the query.

This asymmetry means that threshold_SMC may return a result for one participant but not the other, e.g., P_1 might receive *reject* because t_2 is too low while P_2 receives the actual answer because t_1 is sufficiently high. Nonetheless each P_i's threshold will be respected.

4.5 SMC belief tracking: real world

Lacking a trusted third party in the real world, the participants can use secure multi-party computation and some standard cryptographic techniques to implement P_T's functionality amongst themselves. There are two aspects of P_T that they need to handle: the computation P_T performs, and the hidden state P_T possesses in between queries.

The first aspect is exactly what SMC is designed to do. For the second aspect, we need a way for the participants to maintain P_T's state amongst themselves while preserving its secrecy. (Since we are using the semi-honest adversary model, we do not concern ourselves with integrity; in the malicious setting, standard techniques could be used to enforce integrity.) This state, initially constructed by init_SMC (Figure 9) consists of (1) the parties' secrets, encoded as a state σ_s; (2) policy thresholds t_i; and (3) the current beliefs δ_i. We will refer to this state as Σ_T.[4] We assume Σ_T can be encoded by a binary string of length (exactly) ℓ, for some known ℓ.

The initialization procedure formulated in the idealized world does not output anything to the participants. In the real world, however, the secure computation of init_SMC returns *secret shares* of Σ_T to the parties. That is, the secure computation implements the following (randomized) function after computing Σ_T: choose random $c_1, \ldots, c_{N-1} \in \{0,1\}^\ell$ and set $c_N = \Sigma_T \oplus \left(\bigoplus_{i=1}^{N-1} c_i\right)$. Then each party P_i is given c_i.

The query-evaluation procedure threshold_SMC receives $(c_1, ..., c_N)$ along with the query Q. The procedure begins by reconstructing $\Sigma_T = \bigoplus_{i=1}^{N} c_i$, and then proceeds as usual. Upon completion, threshold_SMC computes (new) shares $c'_1, \ldots, c'_N$ of Σ'_T (as before), and gives c'_i to P_i along with the actual output. (At this point, each P_i can erase the old share c_i.)

Note that each time the sharing is done, nothing additional about Σ_T is revealed from any individual fragment ("share") c_i. (Indeed, each c_i is simply a uniform binary string of length ℓ.) In particular, just as in the ideal world, P_i does not learn whether its policy rejected another participant P_j.

Remark 5. Honest (but curious) participants can derive exactly the same knowledge about each other's secrets from the real-world SMC implementation of P_T that they do from interacting with P_T in the idealized world. Specifically, P_T reveals only the following to each agent P_i in the ideal world:

- Output of a query, if policy checks on δ_i succeed, or
- rejection, if a policy fails.

4.6 Soundness of SMC belief tracking

Suppose that no dishonest parties are detected during the runs of SMC belief tracking. Then, by Remark 5, we can justify soundness in the real world by considering the approach in the idealized world.

Lemma 6. *Suppose principals $P_1, ..., P_N$ wish to execute a query Q. The secret state $\sigma_s = \{x_1 = s_1, ..., x_N = s_N\}$ contains all their secrets. Each has a public threshold t_i for their policy check. Assume the following for each P_i:*

1. *P_i has a belief δ_i about the secret variables.*
2. *P_i's belief δ_i is consistent with σ_s, that is, $\delta_i(\sigma_s) > 0$.*

Suppose $[\![Q]\!]\sigma_s = \sigma'_s$ such that $\sigma'_s(out) = o$. That is, the actual output of the query Q is o. Then, for each agent P_i:

- *If P_i receives output o from P_T, its revised belief is $\delta_i \,||\, (out = o)$.*
- *If P_i is rejected, its belief does not change.*

Specifically, in either case, the procedure threshold_SMC *maintains the correct beliefs.*

Proof. The proof of this lemma reasons similarly Lemma 4: rejection reveals nothing new, and acceptance tracks beliefs precisely. We can see from line 4 of Figure 9, that the procedure used to determine whether P_j will receive an answer or rejection depends on four things:

- The query Q, which is assumed to be public, and chosen independently of secrets.
- P_j's belief, δ_j, about the secrets. This is naturally known by P_j.
- Thresholds t_i for $i \neq j$. These are also assumed to be publicly known.
- Variables x_i, which is just the names of the various secret variables, also known by all.

Since P_j knows all these things, he could determine himself whether P_T will reject him or not. Hence a rejection reveals nothing. In the case P_j receives an answer, we first note the acceptance itself reveals nothing due to the previous argument, and then further, that its belief changes to $\delta_i \,||\, (out = o)$ as claimed. This is due to Theorem 2, as P_j here too is provided only the output of the query.

Note that the condition of consistency in the lemma is only required for the revision operation in the conclusion to be defined. □

The lemma itself is only useful, however, when its premises hold. Specifically, we require P_T to possess the actual beliefs of the participants to start with. This, in turn, means that the initial init_SMC procedure produced them. How the participants arrived at δ, the common belief about the secret variables, used by init_SMC to compute δ_j, is beyond the scope of this work. Once the premises hold, however, Lemma 6 states that they will continue to hold; the tracked beliefs will remain correct and thus the protections of the threshold policies will be maintained.

5. Discussion and experiments

The belief set method and the SMC belief tracking methods present an interesting tradeoff. On the one hand, SMC belief tracking is clearly more precise than belief sets for the simple reason that P_i's estimate of the gain in the other principals' beliefs can consider their secret values exactly without fear that rejection will reveal any information.

On the other hand, SMC belief tracking has two drawbacks. First, the estimate δ_i of what the other parties believe about P_i's secret must be kept hidden from P_i to avoid information leaks. This is unsatisfying from a usability point of view: P_i can be sure that its threshold is not exceeded but cannot see exactly what others know at any point in time. Second, while the performance of SMC has improved quite

[4] Technically, secrets s_i and thresholds t_i could be provided to each invocation of threshold_SMC; we consider them part of the state to emphasize that they are set in place initially, and assumed to not change.

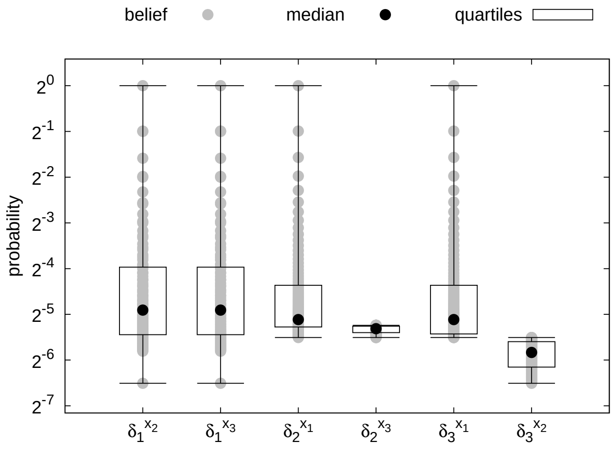


Figure 10. Running example Q_1; plot of max beliefs

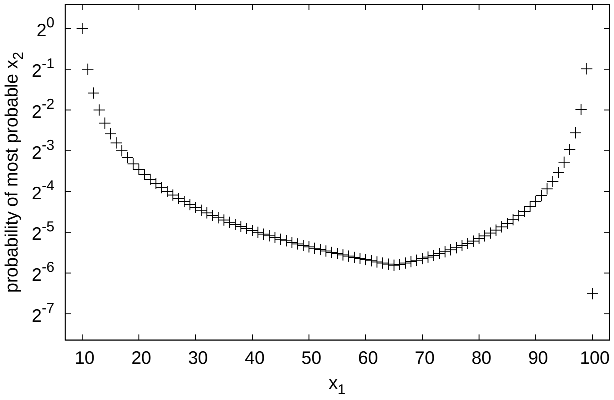


Figure 11. Running example Q_1; $\delta_1^{x_2}$; plot of P_1's max belief about x_2 vs. values of x_1

a bit over time [8], computing a query Q via SMC is still orders of magnitude slower than computing it directly. The belief tracking computation of Q as we have previously implemented it [11] is already orders of magnitude slower than computing Q on the actual values, so performing this computation as an SMC will be significantly slower still. Worse, belief tracking is a recursive procedure, since it is an interpreter, and recursive procedures are hard to implement with SMC. So it remains to be seen whether SMC belief tracking can be implemented in a practical sense.

The belief set method has more hope of seeing a realistic implementation, essentially as an extension of our prior implementation, which is based on abstract interpretation [11]. In our approach, we model a distribution as a set of *probabilistic polyhedra*, which can be thought of as a set of shapes with probabilities attached to them. For example, we could represent that x_1 is uniform distributed in $\{1, \ldots, 10\}$ as the singleton set $\{(1 \leq x_1 \leq 10,\ 1/10)\}$. To improve performance at the cost of precision, we permit *abstracting* these sets; we elide the details due to space constraints. We can very easily produce a naive implementation of belief set tracking (and indeed, have done so), by simply enumerating each of the beliefs $\delta \in \Delta$ when computing $[\![Q]\!]\Delta$, and combining the results. We believe we could extend our abstraction to compute with the Δ directly, and reasonably efficiently.

As a step towards a more thorough evaluation of the precision/performance tradeoff, the remainder of this section compares the precision of the belief set and SMC belief tracking methods on three simple queries. We simulate the SMC belief tracking computation by running our normal implementation in the ideal world setup. We find that SMC belief tracking can be significantly more precise than belief sets, but that belief sets can nevertheless be useful.

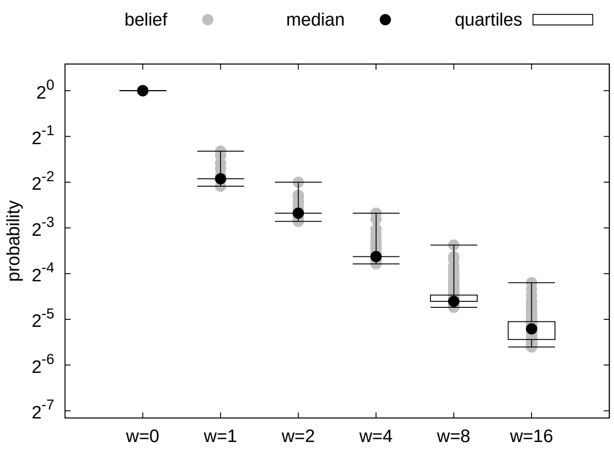
"Am I the richest?" example (Q_1). Consider the running example query Q_1. If all the principals were to evaluate threshold policies to determine the safety of Q_1, they would reason about possible revised beliefs of the participants, where the possibilities vary in their valuation of those participants' secret values, as is described in Section 4.2. If the principals perform this policy check via SMC, they would do so for only one of those possible valuations.

We can better understand the relationship between the two approaches by looking at the range of possible revised beliefs achievable. For some secret values, a principal might learn little; for others, they might learn a lot. We measure this range in terms of the probability of the most probable secret in a principal's belief, for a given valuation of their own secret.

Figure 10 demonstrates the situation for the running example, query Q_1, starting from the initial belief δ uniformly distributing values in $10 \leq x_1, x_2, x_3 \leq 100$. There are 6 relationships considered, for each principal P_i, during their policy decision to allow P_j, with $j \neq i$, to see the query output, they would compute P_j's potential belief about x_i (labeled $\delta_j^{x_i}$ in the figures). These beliefs depend on x_j; the figure shows the potential belief for every possible x_j, the median belief achievable over them as well as the 1st and 3rd quartiles, showing the range of P_j's likely knowledge.[5]

Figure 11 focuses on the first column of Figure 10, $\delta_1^{x_2}$, showing P_1's knowledge about x_2, depending on the value of x_1. At the very top, the most P_1 could learn is when $x_1 = 10$ and the query returns $true$, meaning P_1 was the richest, with the smallest amount of wealth. This lets P_1 conclude that $x_2 = 10$ and $x_3 = 10$. P_1's potential knowledge of x_2 decreases as P_1's wealth grows, up to $x_1 = 65$. At 65, if P_1 is the richest, it is able to narrow x_2 down to 56 values (10 through 65). Starting with $x_1 = 66$, however, P_1 can

[5] It is important to mention that our probabilities are sometimes not exact due to the limitations of the implementation used for the analysis. The true probabilities, however, cannot be larger than those presented here. This imprecision is the reason why belief sets representing P_2 and P_3's beliefs about each other, or about P_1, in Figure 10 appear different, though in actuality they are the same.



```
$similar_w \overset{\text{def}}{=}$ $avg := \frac{x_1+x_2+x_3}{3}$
    if $|x_1 - avg| \leq w \wedge$
       $|x_2 - avg| \leq w \wedge$
       $|x_3 - avg| \leq w$
       then $out := \text{True}$
       else $out := \text{False}$
```

Figure 12. $similar_w$ example; plot of max belief for a variety of windows w sizes

learn more if the query returns $false$, stating that either P_2 or P_3 is richer than P_1. Further increase in x_1, increases its potential knowledge of x_2, culminating at $x_1 = 99$ which lets P_1 conclude that $x_2 = 100$ with a probability close to 0.5. At $x_1 = 100$, the query can only return $true$, hence P_1 learns nothing, keeping its knowledge of x_2 unchanged at $1/91$.

We see that for this query, the belief sets approach would conservatively conclude that all participants could learn x_1 exactly, and that P_1 could learn x_2 and x_3 exactly. On the other hand, it is impossible for P_2 and P_3 to learn each other's values to any confidence.

The benefit of the SMC approach to policy enforcement is that it is free from the overly conservative view of the belief sets approach. In 75% (observing the upper extent of the quartile boxes) or more of the situations, the actual beliefs of the participants do not exceed probability of $2^{-4} \approx 0.06$, which is comparable to the $\frac{1}{91} \approx 0.01$ probability each agent started with. In terms of utility, if the participants set their policy thresholds to as little as 0.06, their policies would allow Q_1 in most cases. The belief sets approach would reject Q_1 for all $t_i < 1$.

Not all queries are pathological for the belief sets approach. We next look at a parameterized query that offers a security vs. utility tradeoff.

"Similar" example The query $similar_w$, depicted at the bottom of Figure 12, determines whether each principal's secret is within w of the average. The choice of window size w determines how much the principals can learn.[6] The plot at the top of the figure shows the possible beliefs after evaluating $similar_w$ with a variety of window values w, with the initial assumption that all values x_i are in uniformly distributed in $1 \leq x_i \leq 100$ (so each of the 100 possibilities has probability 0.01). The scenario is thus completely symmetric in respect to the agents, hence only one of the beliefs is shown in the figure.

When $w = 0$, the query can be completely revealing, as when it returns true, all secrets are equal. Relaxing the window reduces how much each agent can learn. At $w = 2$, each agent, in the worst case, learns every other principal's secret with confidence of 0.25. This worst case already allows non-trivial threshold policies. Going further, with the window set to 16, the query becomes barely revealing, resulting in confidence never reaching over 0.05, comparable to the initial 0.01. Further increase of w can make the query even less revealing to the point of not releasing any information at all, though of course also not providing any utility.

"Millionaires" example. The common motivating example for SMC is a variant of Q_1 involving a group of millionaires wishing to determine which of them is the richest, without revealing their exact worth. The query $richest_p$, given at the bottom of Figure 13, accomplishes this goal, determining which of 3 participants is (strictly) richer than the other two. An addition to this query has been made to provide a means of injecting noise into its answers to limit potential knowledge gain.

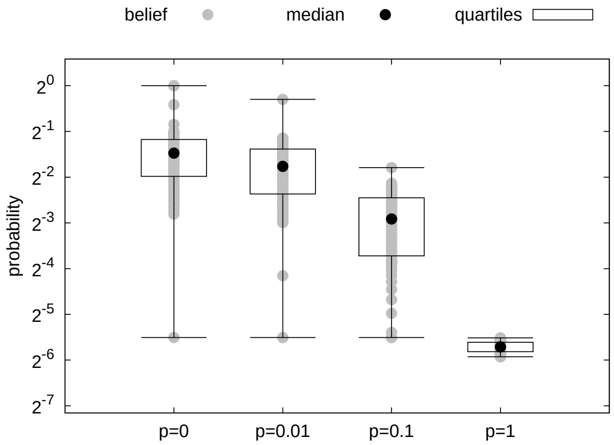
The output $out = 0$ designates that none of the three were strictly richest. The query is concluded with a step that noises the result. The assignment $out := uniform\,\{0, 1, 2, 3\}$ is shorthand notation for a series of pif statements, whose effect is to set out to one of the 4 values, with uniform probability. Thus given the parameter p, this query will just randomly return, with probability p, one of the 4 possible outputs.

A significant benefit of using probabilistic programming languages is that the effect of non-determinism described using these probabilistic statements is taken into account; we can determine exactly what a rational agent would conclude from learning the output of such a noised query.

Figure 13 summarizes the beliefs of every agent assuming initially each is equally likely to be worth between \$10 and \$100. This scenario is also symmetric hence only the belief of one agent about a single other agent is shown.

With $p = 0$, that is, no chance of random output, the query is potentially fully revealing, but in most cases still keeping the participants below 0.5 certainty. With a 0.01 chance of random output, the worst case no longer results in

[6] Note that the language described in Figure 2 cannot directly express this query due to the lack of the division and absolute value operations. However, we can achieve the same effect by multiplying all expressions by 3 and replacing the conditions on absolute values by a pairs of upper and lower bounding conditions.



```
$richest_p \overset{\text{def}}{=}$  $out := 0$
             if $x_1 > x_2 \wedge x_1 > x_3$ then $out := 1$
             if $x_2 > x_1 \wedge x_2 > x_3$ then $out := 2$
             if $x_3 > x_1 \wedge x_3 > x_2$ then $out := 3$
             pif $p$ then $out := uniform\{0, 1, 2, 3\}$
```

Figure 13. $richest_p$ example; plot of max belief for a variety of noising probabilities p

absolute certainty, though close to it. Randomizing the output with $p = 0.1$, keeps the agents' certainty almost below 0.5 in the worst case. Getting closer to $p = 1$ the beliefs approach the initial ones. At $p = 1$ the query reveals nothing, though our approximate implementation does produce some variation in the upper bound.

6. Related Work

Almost all prior work on SMC treats the function f being computed by the parties as given, and is unconcerned with the question of whether the parties should agree to compute f in the first place. The only exceptions we are aware of are two papers [4, 1] that consider SMC in conjunction with *differential privacy* [5, 3]. Dwork et al. [4] show that if f is a differentially private function, then the process of running an SMC protocol that computes f is also differentially private (at least in a computational sense). Beimel et al. [1] observe that if the end goal is a distributed protocol that is differentially private, then SMC may be overkill and more efficient alternatives may be possible.

The security goal we are aiming for is incomparable with that of differential privacy. Moreover, in contrast to above-mentioned work, our determination of whether a function f is "safe" to compute will explicitly depend on the parties' actual inputs as well as any (known or assumed) prior knowledge that parties have about each others' inputs.

7. Conclusions

In this paper we have presented two methods that apply *knowledge-based security policies* to the problem of determining whether participating in a secure multiparty computation could unsafely reveal too much about a participant's secret input. Ours are the first techniques that consider the actual secrets and prior knowledge of participants (potentially gained from previous SMC's) when making this determination, making our approach more permissive (in accepting more functions), and potentially safer, than techniques that disregard this information. Experiments with the two methods show that the *SMC belief tracking* method is the more permissive of the two, but it remains to be seen whether this method can be implemented efficiently.

References

[1] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In *Advances in Cryptology — Crypto 2008*, volume 5157 of *LNCS*, pages 451–468. Springer, 2008.

[2] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *J. Comput. Secur.*, 17(5), 2009.

[3] Cynthia Dwork. Differential privacy. In *33rd Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

[4] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology — Eurocrypt 2006*, volume 4004 of *LNCS*, pages 486–503. Springer, 2006.

[5] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *3rd Theory of Cryptography Conference — TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, 2006.

[6] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.

[7] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[8] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security*, 2011.

[9] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. Simulatable auditing. In *PODS*, 2005.

[10] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *J. Privacy and Confidentiality*, 1(1):59–98, 2009.

[11] Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. Dynamic enforcement of knowledge-based security policies. In *Proceedings of the Computer Security Foundations Symposium (CSF)*, June 2011.

[12] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.