Secure Collaboration in Engineering Systems Design

Shumiao Wang
Currently at
Google Inc.

Siddharth Bhandari
Currently at
Tata Institute of Fundamental Research

Siva Chaitanya Chaduvula
School of Mechanical Engineering
Purdue University

Mikhail J. Atallah
Department of Computer Science
Purdue University

Jitesh H. Panchal[*]
School of Mechanical Engineering
Purdue University

Karthik Ramani
School of Mechanical Engineering
Purdue University

ABSTRACT

The goal in this paper is to enable collaboration in the co-design of engineering artifacts when participants are reluctant to share their design-related confidential and proprietary information with other co-designers, even though such information is needed to analyze and validate the overall design. We demonstrate the viability of co-design by multiple entities who view the parameters of their contributions to the joint design to be confidential. In addition to satisfying this confidentiality requirement, an online co-design process must result in a design that is of the same quality as if full sharing of information had taken place between the co-designers. *We present online co-design protocols that satisfy both requirements, and demonstrate their practicality using a simple example of co-design of an automotive suspension system and the tires. Our protocols do not use any cryptographic primitives – they only use the kinds of mathematical operations that are currently used in single-designer situations. The participants in the online design protocols include the co-designers, and a cloud server that facilitates the process while learning nothing about the participants' confidential information or about the characteristics of the co-designed system. The only assumption made about this cloud server is that it does not collude with some participants against other participants. We do* not *assume that the server does not, on its own, attempt to compute as much*

[*]Email: panchal@purdue.edu Address all correspondence to this author.

information as it can about the confidential inputs and outputs of the co-design process: It can *make a transcript of the protocol and later attempt to infer all possible information from it, so it is a feature of our protocols the cloud server can infer nothing from such a transcript.*

1 Introduction

1.1 Need for Secure Collaboration in Engineering Systems Design

While online collaborations are revolutionizing design and manufacturing, they are falling short of achieving their full potential because there are significant barriers to information-sharing. These include the fear that information voluntarily shared with a partner can later be used in ways contrary to the volunteer's interests, the fear that sensitive information will leak to a competitor [1]. What a participant gains from the mutual sharing of information can be more than offset by what is lost from revealing confidential and proprietary information. There are also government regulations about information-sharing: if one of the parties is government, or if foreign entities are among the participants, then there are national security reasons to protect sensitive information. Within engineering design, such privacy concerns are the key barriers impeding collaboration [2, 3], specifically, in three design scenarios: collaborative exploration of design space, crowdsourcing, and cloud-based design and manufacturing.

During the early stages of systems design, before parties have decided whether the collaboration would be meaningful, designers generally need to exchange information about capabilities and sub-system behaviors. Such information can either be in terms of performance curves and datasheets, or test results from independent entities. For complex systems, designers may need detailed mathematical models encapsulating the behavior to ensure that the sub-system performs well within the context of the overall system. Such behavioral models embody significant knowledge, and have the potential to reveal confidential information about the subsystem. In such cases, designers may be reluctant to share the models for collaborative exploration of design spaces. Existing research on collaborative engineering design has not addressed such scenarios because of the implicit assumption that collaborating entities are willing to share all the details about their part of the design. Collaborative design platforms such as the product data management (PDM) and product lifecycle management (PLM) systems are only focused on access management, i.e., a participant either has access to data/models or not. These access control approaches [4,5] are designed against malicious agents outside the collaboration. However, they are not useful against malicious or curious partners. Moreover, such an approach works well for later stages of design, after partnerships have been established, but not for the early stages of design described above. Similarly, distributed collaborative simulation platforms such as iSIGHT, ModelCenter, and FIPER [6] also lack the capabilities to protect confidential information from prospective partners, while enabling joint simulation.

Crowdsourcing [7] is another scenario where mutually beneficial design interactions may be hampered due to concerns over security of private information. Crowdsourcing is the practice of outsourcing tasks, traditionally performed by employees or suppliers, to a large group of people in the form of open tournaments [7]. It involves two types of parties – seekers and solvers. Seekers have a problem that they want to solve, and solvers are individuals who compete to solve the problems. The solver who submits the best solution receives a pre-specified award. Crowdsourcing is attractive for product development organizations because of various benefits, including access to a broad pool of solvers, increased capacity of idea generation through open innovation, and the opportunity for cost savings because the payment is made only after solutions are obtained [8]. However, the downside of crowdsourcing is that solvers may be unwilling to respond to the competition due to the fear that their ideas may be misused without a reward. Similarly, if the seeker is a defense agency, it may be unwilling to share the details of their problem in online settings for security reasons. Hence, the success of crowdsourcing as a mechanism for design collaboration significantly depends on the availability of secure collaboration in engineering design.

The third emerging scenario where secure collaboration is important is cloud-based design and manufacturing (CBDM). CBDM is a product realization model that enables rapid product development through on-demand distributed system of interconnected physical and virtual design and manufacturing services [9]. The economics of using clouds for designing by services [10] are compelling due to the potential for dynamically adapting the amount of computing resources and hardware needed. However, security and privacy are some of the main factors affecting the adoption of public clouds by organizations [11]. Because of the trust issues, many organizations prefer secure and private clouds that are more expensive to develop, operate and maintain.

As a summary, there are three main impediments to design collaboration, which have resulted in the unwillingness to: a) collaborate during design exploration, and b) participate in crowdsourcing tournaments, and c) make use of cloud. The goal of the work presented in this paper is to make such online design collaborations possible in the co-design of engineering artifacts when participants are reluctant to share their own confidential and proprietary information with the other co-designers, even though such information is needed for producing a design of the appropriate quality and performance. Our main results are online protocols that enable the parties in the co-design to cooperatively achieve the desired result without any of them revealing their own private data, even though the jointly designed engineering artifact depends crucially on every participant's private data. We also demonstrate the viability of our protocols by implementing them and experimentally evaluating them.

1.2 Uniqueness and Contributions

Our approach does not use encryption and other cryptographic primitives, even though it is well known that techniques from the cryptographic area of secure multi-party computation [12] are perfectly capable, in theory, of solving all of the co-design problems considered in this paper. This is primarily due to performance issues. The use of cryptographic techniques would involve operations like fully homomorphic encryption [13] and secure circuit evaluation [14] that, when used in the context of the mathematical calculations for engineering design, would be both very slow and very hard to implement, which is also observed in [15]. In contrast, our protocols use the same mathematical primitives that arise anyway in the design process.

On the other hand, our approach does increase the size of the numbers involved, but the addition and multiplication of large numbers is nevertheless orders of magnitude faster than the use of the above-mentioned cryptographic primitives. The need for using larger numbers than those in the current design process, arises because in our protocols co-designers hide their numbers through adding and multiplying them with randoms, and it is well known that this does a good job of hiding only if the random numbers used are much larger than the numbers they are meant to hide (unless the arithmetic is modular, in which case the hiding is perfect). Using modular arithmetic (for perfect hiding) is something worth investigating in follow-on work – we have chosen to avoid it in our current design and implementation because (i) it would substantially complicate our protocols (and may even make some of them impractical); and (ii) our current use of large randoms for hiding (without modular arithmetic) does a good enough job for most practical situations. In addition to the protocols we designed and implemented for the specific co-design scenarios we considered, and our demonstration of their practical viability, another contribution of this work is that the computational building blocks we presented and used in our protocols are highly likely to be useful for other scenarios and in other contexts.

The outline of the paper is as follows. In the following section, we discuss engineering design scenarios where secure collaboration can be utilized. Section 3 provides details of the protocols used for secure computation in design. A discussion of the characteristics of the protocols is provided in Section 4. The implementation details and results for an illustrative design scenario are presented in Section 5. Finally, closing comments are presented in Section 6.

2 Co-Design Scenario for Secure Collaboration

2.1 Secure Collaboration in Designing Composable Dynamical Systems

Consider a scenario shown in Figure 1 where two parties, Alice and Bob, own subsystems 1 and 2 respectively. They possess detailed information about the behaviors of these subsystems. Specifically, we focus on the dynamic characteristics of these subsystems, represented by state-space equations. The behaviors of the individual subsystems can be represented using matrices A_i, B_i, C_i, and D_i, and the behavior of the composed system is described using matrices A, B, C, and D. Matrices A, B, C, and D are dependent on A_i, B_i, C_i, and D_i. We assume that Alice and Bob are both interested in knowing the behavior of the composed system but are unwilling to share the details of their matrices with anyone (i.e., neither with each other, nor with any third party). It is also assumed that both the parties have no incentive to lie about their individual systems properties. This is a reasonable assumption in design exploration to establish collaboration because once the parties agree to collaborate, they will share the details of the models with each other for detailed design.

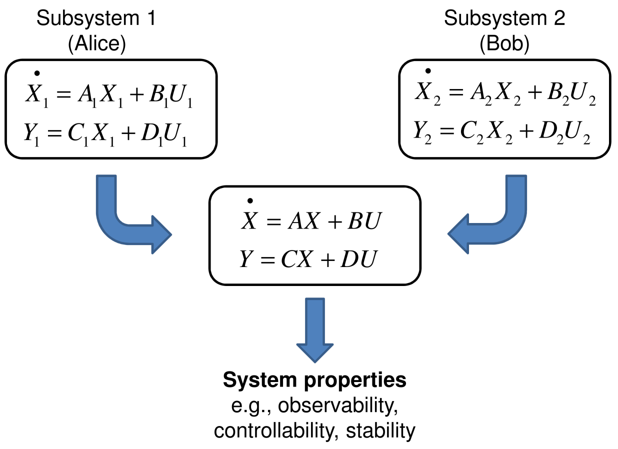

Fig. 1. Co-design scenario

The state-space formulation of dynamical systems is chosen because a) any dynamical system, including those with multiple inputs and multiple outputs, can be represented in terms of the state-space equations, b) it is general enough to be applied to mechanical and non-mechanical systems, c) non-linear systems can be modeled by linearizing the system about the operating point, and d) several system-level properties such as observability, controllability, and stability can be revealed without completely solving the state-space equations.

For privacy preservation, there are restrictions on the kinds of system-level properties which can be explored. For example, if the parties want to know the detailed responses for different types of inputs, each party may be able to extract the hidden parameters (or other related information such as bounds on parameters) just based on the responses. If the properties under investigation implicitly reveal any information about the proprietary data, the privacy constraint is violated. Hence, such properties cannot be jointly studied. However, properties which do not implicitly reveal confidential information can be investigated, and protocols for those properties may be developed. Examples of such properties include observability, controllability, Lyapunov stability, detectability, and stabilizability. In this paper we study the first three properties. In addition to this the protocols presented in this paper can be used to answer questions such as “does the solution enter a certain domain?” without revealing any additional information about the private data.

The level of required privacy can be categorized into two scenarios:

1. A scenario where no single party has knowledge of the structure of the composed system, i.e., how the system-level matrices depend on the sub-system-level matrices. The parties are not willing to share the details of their respective matrices with each other or with any third party.
2. A scenario where the mathematical structure of the composed system (i.e., how the system-level matrices depend on the sub-system level matrices) is known to at least one of the collaborating parties while the parameter values of the various components in the system are private.

In this paper, we consider linear time-invariant (LTI) systems designed under the second scenario where the structure of the collaborative system is known to at least one party with the parameter values being proprietary to the respective parties. For this scenario, we present protocols to jointly evaluate properties of the composed dynamical system.

2.2 Illustrative Example: Co-Design of an Automotive Suspension System and Tires

We illustrate the protocols using a specific example of an automotive company (Alice) collaborating with a tire company (Bob). Alice has a model of the dynamics of the suspension system, which is assumed to be a moving vehicle and suspension system half-car model [16] (see Figure 2). Bob has a model of the tires, which is assumed to be a simple spring-mass system. It is also assumed that Alice has complete knowledge of the structure of the composed system but does not know the parameter values in the tire model which are Bob’s proprietary data. Such an assumption is justified for systems that are mature and their topologies are well known. In this case, the assumption is that the tire model is well known whereas the suspension-system model may have a proprietary structure. Through the collaboration, both parties are interested in learning about controllability and observability of the co-designed system.

As shown in Figure 2, Alice’s private inputs to the collaborative system are M, I, K_{fs}, K_{rs}, b_f, b_r (and distances not

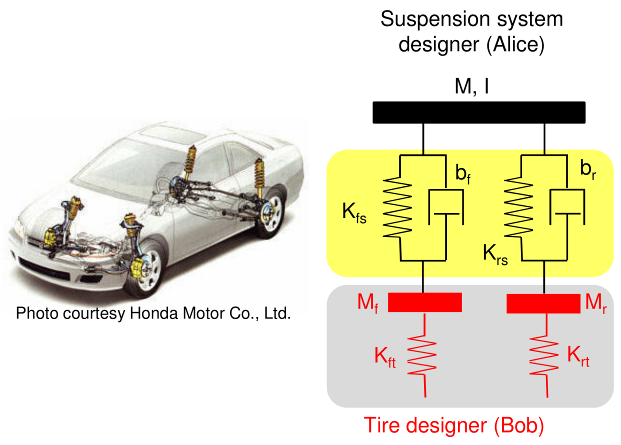

Fig. 2. Collaborative half-car suspension system model

shown in the figure, L_f and L_r, which are distances of front and rear tires from the center of mass of the car). Bob's private inputs are M_f, M_r, K_{ft}, K_{rt}. The matrices representing the collaborative system in the state-space [16] model are:

$$A^1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -(K_{fs}+K_{rs})/M & -(B_f+B_r)/M & K_{fs}/M & B_f/M \\ K_{rs}/M & B_r/M & (K_{rs}L_r-K_{fs}L_f)/M & (B_rL_r-B_fL_f)/M \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ K_{fs}/M_f & B_f/M_f & -(K_{fs}+K_{ft})/M_f & -B_f/M_f \\ 0 & 0 & K_{rs}L_f/M_f & B_fL_f/M_f \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ K_{rs}/M_r & B_r/M_r & 0 & 0 \\ -(K_{rs}+K_{rt})/M_r & -B_r/M_r & -K_{rs}L_r/M_r & -B_rL_r/M_r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ (K_{rs}L_r-K_{fs}L_f)/I & (B_rL_r-B_fL_f)/I & K_{rs}L_f/I & B_fL_f/I \\ -K_{rs}L_r/I & -B_rL_r/I & -(K_{fs}L_f^2+K_{rs}L_r^2)/I & -(B_fL_f^2+B_rL_r^2)/I \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_{ft}/M_f & 0 \\ 0 & 0 \\ 0 & K_{rt}/M_r \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

and

$$C = \begin{bmatrix} 1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,0 \end{bmatrix}$$

The full state controllability and observability are given as [17]:

$$Controllability = \begin{bmatrix} B \mid AB \mid \ldots \mid A^7 B \end{bmatrix}$$

and

$$Oberservability = \begin{bmatrix} C \\ CA \\ \ldots \\ \ldots \\ CA^7 \end{bmatrix}$$

The models considered in this paper are simplistic compared to those used in practice, because our aim is to lay the foundational framework for such collaborations without getting distracted by the modeling details. Hence, considering a

[1]A is an 8×8 matrix, and due to the space limit, we typed the right half 8×4 submatrix under the left half.

simple model is justified as it highlights the issues and how they are addressed while keeping the complexity down. The protocols presented in the paper are independent of the problem size. Hence, they can directly be used for more complex models. Also, the collaboration here involves only two parties but upon observing the techniques employed we see that modeling such collaborations between multiple parties is straightforward by using the protocols for multiple parties in the collaboration. This is enabled due to the nature of the protocols which can be generalized for any number of collaborating parties.

Our protocols not only preserve the privacy of the proprietary parameters of the respective parties but also ensure that the design of Alice's subsystem is not leaked. To this end our protocols run in two stages (see Figure 3):

1. The additive split protocol (described in Section 3.2) splits the terms of the state-space matrices (A, B, C, and D) among the collaborating parties. Note that these matrices are functions of the private parameters of the respective parties and Alice has information about the structure of the overall system, i.e., how the elements of the matrices are dependent on the privately held parameters.
2. The next set of protocols enables us to perform various mathematical operations on the obtained additively split terms. We check the full rank of the observability and controllability matrices and check for the negative definiteness of the matrix A (using protocols in Section 3.3).

We discuss the protocols in detail in the following section.

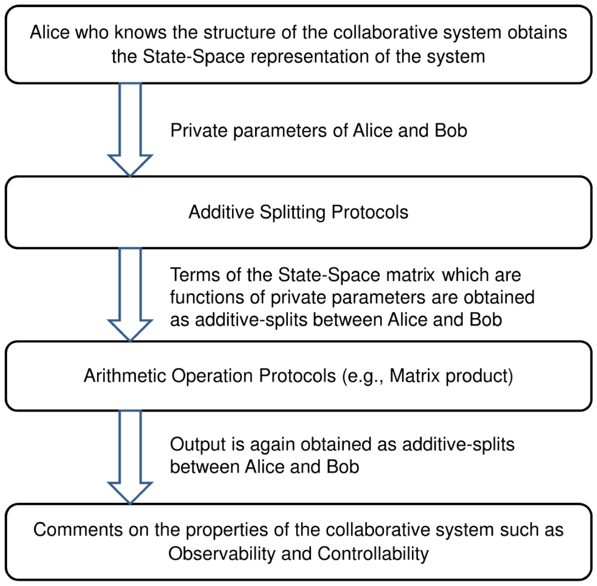
Alice who knows the structure of the collaborative system obtains the State-Space representation of the system
Private parameters of Alice and Bob
Additive Splitting Protocols
Terms of the State-Space matrix which are functions of private parameters are obtained as additive-splits between Alice and Bob
Arithmetic Operation Protocols (e.g., Matrix product)
Output is again obtained as additive-splits between Alice and Bob
Comments on the properties of the collaborative system such as Observability and Controllability

Fig. 3. High-level overview of the protocol

3 Protocols for Secure Co-Design

In this section, three sets of protocols are presented. The general problem setting is that two parties Alice and Bob, denoted by A and B, both possess some private input data, and want to perform specific computations with help from a cloud server S, without leaking his/her input to the other party or to S. The server S is modeled as a honest-but-curious server, i.e., S behaves as the protocol states and does not collude with either client, but tries to learn any confidential information about the clients' input or output. Our security assumption is that adding a large random to a value hides it and multiplying a nonzero value by a large random hides it. We will discuss this in detail in Section 4.

Therefore, we use an additively split scheme to hide all the values involved. To **additively split** (or **split** for short) a value v into two, say v_A and v_B, a value is randomly chosen as v_A, and $v_B = (v - v_A)$. Here, v_A and v_B are called the **secret shares** of v. If two parties each have one share of v, we say they **additively share** (or **share** for short) the value v. A party possessing either v_A or v_B could not infer v from that, and the statistical information they obtain is negligible if the random numbers used for hiding are much larger than the values they are used to hide (more on this in Section 4).

In Section 3.1, we present a set of needed arithmetic protocols. All the input values to the protocols are additively shared by A and B, and these protocols enable them to perform arithmetic operations and get the results also in the split form. The ingenuinity of these protocols is to enable A and B to compute without allowing server to learn. A and B achieve this by morphing the additive splits (of inputs) before sending to the server. This morphing function need to have the following properties:

1. It should be possible to recreate the morphing by both A and B.
2. Removal of morphing should be possible on the additive splits (of computed result) received from the server .

After each protocol, we briefly show its correctness and prove that no party could learn any information other than its own input during executing the current protocol.

In most real world applications, the inputs to the co-design system are not initially additively split and shared by the two co-designers, and moreover, the designers want to know the true result, not just a random-like share. Therefore, to make these protocols complete, a general method is provided in Section 3.2 to additively split the inputs between the designers, and a protocol is presented to enable the designers to merge the output back from its split form.

With the ability to split the inputs and merge the outputs, we can solve complicated co-design problems by decomposing them into those building blocks in Section 3.1. Section 3.3 presents several high level protocols to test specific properties of a matrix, which suffice to solve the problems presented in Section 2, just as examples of decomposing problems.

3.1 Building Blocks

In this subsection, we present a set of protocols to achieve arithmetic computations on inputs shared by two parties A and B, without leaking one party's input to the other or to S, which is proved in their following security analysis. These protocols are building blocks for the customized high-level protocols.

A and B are able to generate the same random numbers without communication with each other. This is achieved in the set-up phase by agreeing on a secret random seed and a cryptographically strong hash function. Using the same hash chain and the same initial seed, A and B generate the random numbers at their own machine. Such random numbers, known to both A and B, are called shared random numbers when presenting the protocols. All the shared random values used in the protocols should be nonzero. If a zero is generated, A and B just discard it and generate another.

3.1.1 Addition and Subtraction Protocol (ASP)

Input: A value (or vector, matrix) v additively shared by A and B as v_A and v_B; and a value (or vector, matrix) u additively shared by A and B as u_A and u_B.

Output: w, additively shared by A and B as w_A and w_B, and $w = v \pm u$.

Protocol:

1. A computes $w_A = v_A \pm u_A$.
2. B computes $w_B = v_B \pm u_B$.

Correctness: It can be easily verified that $w = w_A + w_B = v \pm u$.

Security: A and B only do computation based on their own input, and thus cannot learn any information by performing this protocol.

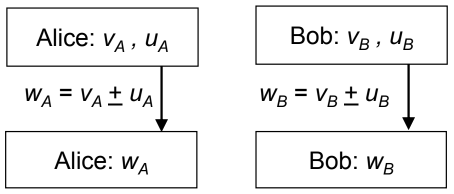

Fig. 4. Flow of the Addition and Subtraction Protocol (ASP)

3.1.2 Multiplication Protocol (MP)

Input: A value v additively shared by A and B as v_A and v_B; and a value u additively shared by A and B as u_A and u_B.

Output: w, additively shared by A and B as w_A and w_B, and $w = vu$.

Protocol:

1. A generates four shared random numbers r_1, r_2, r_3 and r_4, and sends a pair $(r_1 * v_A + r_2, r_3 * u_A + r_4)$ to S. B generates the same shared random numbers r_1, r_2, r_3 and r_4, and sends a pair $(r_1 * v_B - r_2, r_3 * u_B - r_4)$ to S.
2. S receives a pair from A denoted as $(P_{1,A}, P_{2,A})$, and a pair from B denoted as $(P_{1,B}, P_{2,B})$, and computes a value $t = (P_{1,A} + P_{1,B}) * (P_{2,A} + P_{2,B})$. S additively splits t into t_A and t_B, and sends them to A and B respectively.
3. A receives t_A, generates another shared random number r_5, and computes $w_A = t_A/(r_1 r_3) + r_5$.
   B receives t_B, generates the same r_5, and computes $w_B = t_B/(r_1 r_3) - r_5$.

Correctness: This can be verified as $w = w_A + w_B = (t_A + t_B)/(r_1 r_3) = vu$.

Security: From S's view, the inputs are hidden by four random values in step 2 and S cannot learn them. And the output w

is hidden from S by multiplying $(r_1 r_3)$, so S cannot learn it either. However, in step 2, S knows how it splits t, i.e., the ratio between its two shares t_A and t_B. So in step 3, r_5 is used to get a fresh split of w so that the ratio between the two shares of w is not same as t, preventing S from knowing information about how the output is split. From A and B's view, the only values received by them are split shares of a value, which will not leak to them the result or information about how the output is split. Except these values, no party obtains additional information by executing this protocol.

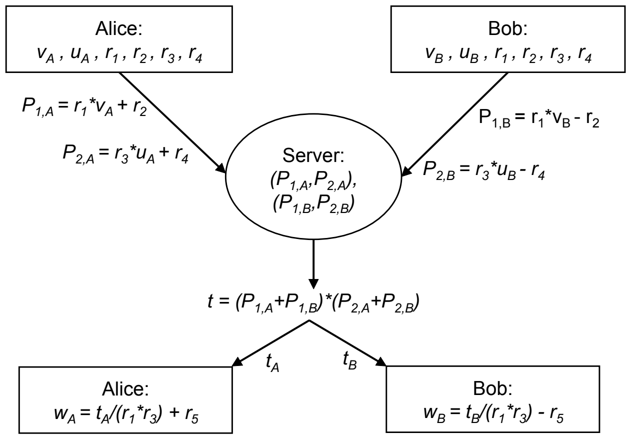
Alice:
v_A , u_A , r_1 , r_2 , r_3 , r_4
Bob:
v_B , u_B , r_1 , r_2 , r_3 , r_4
$P_{1,A} = r_1*v_A + r_2$
$P_{2,A} = r_3*u_A + r_4$
$P_{1,B} = r_1*v_B - r_2$
$P_{2,B} = r_3*u_B - r_4$
Server:
$(P_{1,A},P_{2,A})$,
$(P_{1,B},P_{2,B})$
$t = (P_{1,A}+P_{1,B})*(P_{2,A}+P_{2,B})$
t_A
t_B
Alice:
$w_A = t_A/(r_1*r_3) + r_5$
Bob:
$w_B = t_B/(r_1*r_3) - r_5$

Fig. 5. Flow of the Multiplication Protocol

3.1.3 Division Protocol (DP)

Input: A value v additively shared by A and B as v_A and v_B; and a nonzero value u additively shared by A and B as u_A and u_B.

Output: w, additively shared by A and B as w_A and w_B, and $w = v/u$.

Protocol:

1. A generates four shared random numbers r_1, r_2, r_3 and r_4, and sends a pair $(v_A * r_1 + r_2, u_A * r_3 + r_4)$ to S. B generates the same shared random numbers r_1, r_2, r_3 and r_4, and sends a pair $(v_B * r_1 - r_2, u_B * r_3 - r_4)$ to S.
2. S receives a pair from A denoted as $(P_{1,A}, P_{2,A})$, and a pair from B denoted as $(P_{1,B}, P_{2,B})$, and computes a value $q = (P_{1,A} + P_{1,B})/(P_{2,A} + P_{2,B})$. S additively splits q into q_A and q_B, and sends them to A and B respectively.
3. A receives q_A, generates another shared random number r_5, and computes $w_A = q_A * (r_3/r_1) + r_5$.
   B receives q_B, generates the same r_5, and computes $w_B = q_B * (r_3/r_1) - r_5$.

Correctness: This can be verified as $w = w_A + w_B = q * (r_3/r_1) = (v_A + v_B)/(u_A + u_B) = v/u$.

Security: The analysis of security is similar as the MP protocol.

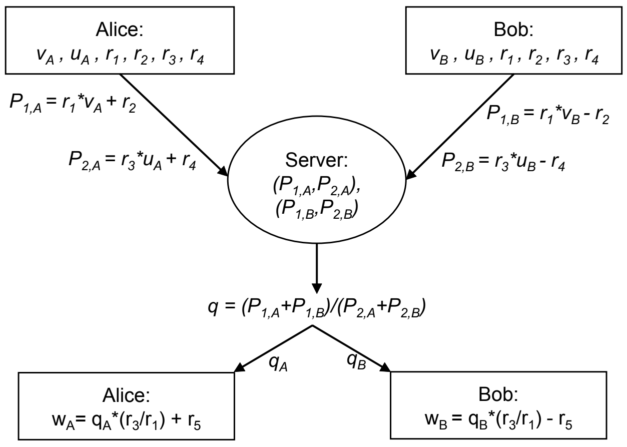
Alice:
v_A , u_A , r_1, r_2, r_3, r_4
Bob:
v_B , u_B , r_1, r_2, r_3, r_4
$P_{1,A} = r_1{*}v_A + r_2$
$P_{2,A} = r_3{*}u_A + r_4$
$P_{1,B} = r_1{*}v_B - r_2$
$P_{2,B} = r_3{*}u_B - r_4$
Server:
$(P_{1,A},P_{2,A})$,
$(P_{1,B},P_{2,B})$
$q = (P_{1,A}+P_{1,B})/(P_{2,A}+P_{2,B})$
q_A
q_B
Alice:
$w_A = q_A{*}(r_3/r_1) + r_5$
Bob:
$w_B = q_B{*}(r_3/r_1) - r_5$

Fig. 6. Flow of the Division Protocol

3.1.4 Logarithm protocol

Input: A value v additively shared by A and B as v_A and v_B.

Output: A complex value x_A held by A and x_B held by B, whereas $x_A + x_B = \ln(v)$.

Protocol:

1. A generates two shared random numbers r_1 and r_2, and sends $v_A * r_1 + r_2$ to S. B generates the same shared random numbers r_1 and r_2, and sends $v_B * r_1 - r_2$ to S.
2. S receives a value from A denoted as p_A, and a value from B denoted as p_B, and computes a complex number $q = \ln(p_A + p_B)$. S additively splits q into two complex values as q_A and q_B, and sends them correspondingly to A and B.
3. A receives q_A from S, generates a shared complex random number r_3, and computes $x_A = q_A + r_3$. B receives q_B from S, generates the same r_3, and computes $x_B = q_B - \ln(r_1) - r_3$.

Correctness: This is verified as $x_A + x_B = q - \ln(r_1) = \ln(p_A + p_B) - \ln(r_1) = \ln((p_A + p_B)/r_1) = \ln(v)$.

Security: From S's view, the inputs are hidden by r_1 and r_2 and S cannot learn them. And the output x is hidden from S by $\ln(r_1)$, so S cannot learn it either. r_3 prevents S from knowing information about how the output is split by q_A and q_B. From A and B's view, the only values received by them are split shares of a value, which will not leak to them the result or information about how the output is split. Except these values, no party obtains additional information by executing this protocol.

3.1.5 Exponentiation Protocol

Input: Assume that A and B mutually agree upon n which is larger than their individual inputs (v, u respectively). Let B represents u as an array of n bits which is denoted by $\mathbf{u}$. Invert every bit in $\mathbf{u}$ and denote this array by $\mathbf{c}$. A determines two arrays of length n: $\mathbf{x} = [v^{2^0}, v^{2^1}, v^{2^2}, v^{2^3}, \dots v^{2^{n-1}}]$ and $\mathbf{y} = [1, 1, 1 \dots 1]$. A and B share one of their additive splits with each other. So, A has $\mathbf{x_A}, \mathbf{u_A}, \mathbf{y_A}, \mathbf{c_A}$ and B has $\mathbf{x_B}, \mathbf{u_B}, \mathbf{y_B}, \mathbf{c_B}$ as inputs to this protocol.

Output: w_n, additively shared by A and B as $w_{A,n}$ and $w_{B.n}$, and $w_n = v^u$.

Protocol: This protocol is split into three phases: In phase 1, A and B determine an intermediate array $\mathbf{t} = \mathbf{u} \circ \mathbf{x}$. Here $\circ$

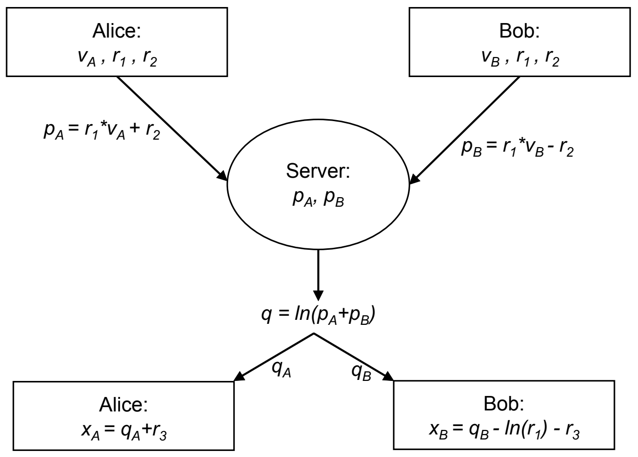

Fig. 7. Flow of Logarithm Protocol

denotes hadamard product and the elements in the array $\mathbf{t}$ are given by $(t_i = u_i x_i \quad \forall i \in (1,n))$. Phase 2 is very similar to Phase 1. In phase 2, A and B determine an intermediate array $\mathbf{m} = \mathbf{y} \circ \mathbf{c}$. In phase 3, A and B add their additive splits obtained from phase 1 and 2 and determine the product of all the elements in it. This product turns out to be equal to v^u.

Phase 1

Input: A has arrays $\mathbf{x_A}, \mathbf{u_A}$ and B has $\mathbf{x_B}, \mathbf{u_B}$

Output: A and B receive $\mathbf{t_A}$, $\mathbf{t_B}$ from Server, where $\mathbf{t_A} + \mathbf{t_B} = \mathbf{x} \circ \mathbf{u}$. Note that $\circ$ stands for hadamard product.

Protocol: A and B perform MP protocol for all the n pairs in the two vectors and thus get n intermediate results. These results are stored in an array $\mathbf{q}$. Figure 8 displays the multiplication for an i^{th} element in $\mathbf{x}, \mathbf{u}$. By the end of n multiplications in Phase 1, Alice and Bob have arrays $\mathbf{t_A}$ and $\mathbf{t_B}$ respectively.

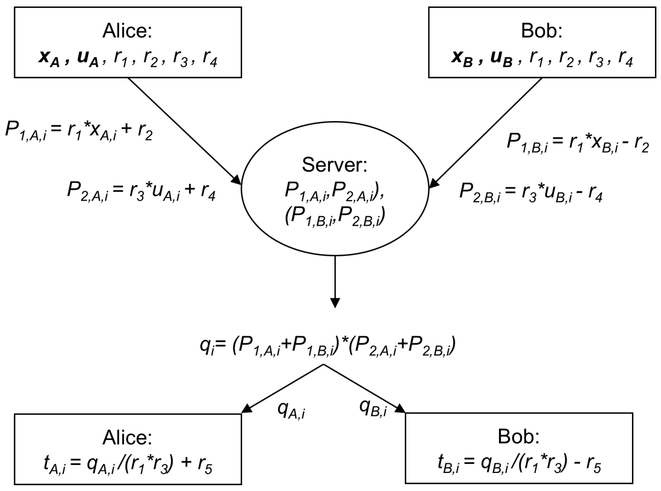
Alice:
$\mathbf{x_A}$, $\mathbf{u_A}$, r_1 , r_2 , r_3 , r_4
Bob:
$\mathbf{x_B}$, $\mathbf{u_B}$, r_1 , r_2 , r_3 , r_4
$P_{1,A,i} = r_1*x_{A,i} + r_2$
$P_{2,A,i} = r_3*u_{A,i} + r_4$
$P_{1,B,i} = r_1*x_{B,i} - r_2$
$P_{2,B,i} = r_3*u_{B,i} - r_4$
Server:
$(P_{1,A,i}, P_{2,A,i})$,
$(P_{1,B,i}, P_{2,B,i})$
$q_i = (P_{1,A,i}+P_{1,B,i})*(P_{2,A,i}+P_{2,B,i})$
$q_{A,i}$
$q_{B,i}$
Alice:
$t_{A,i} = q_{A,i}/(r_1*r_3) + r_5$
Bob:
$t_{B,i} = q_{B,i}/(r_1*r_3) - r_5$

Fig. 8. Flow of Phase 1

Phase 2

Input: A has arrays $\mathbf{y_A}, \mathbf{c_A}$ and B has $\mathbf{y_B}, \mathbf{c_B}$

Output: A and B receive $\mathbf{z_A}$, $\mathbf{z_B}$ from Server, where $\mathbf{z_A} + \mathbf{z_B} = \mathbf{y} \circ \mathbf{c}$. Note that $\circ$ stands for hadamard product.

Protocol: The protocol is same as that stated in Phase 1. Only the inputs are different. Figure 9 displays the multiplication of between i^{th} elements in $\mathbf{y}, \mathbf{c}$. By the end of n multiplications in Phase 2, Alice and Bob have arrays $\mathbf{m_A}$ and $\mathbf{m_B}$ respectively.

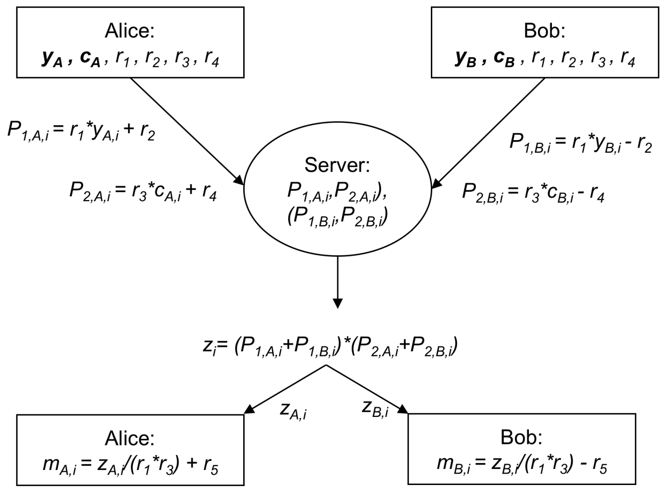
Alice:
y_A , c_A , r_1 , r_2 , r_3 , r_4
Bob:
y_B , c_B , r_1 , r_2 , r_3 , r_4
$P_{1,A,i} = r_1 * y_{A,i} + r_2$
$P_{2,A,i} = r_3 * c_{A,i} + r_4$
$P_{1,B,i} = r_1 * y_{B,i} - r_2$
$P_{2,B,i} = r_3 * c_{B,i} - r_4$
Server:
$(P_{1,A,i}, P_{2,A,i})$,
$(P_{1,B,i}, P_{2,B,i})$
$z_i = (P_{1,A,i} + P_{1,B,i}) * (P_{2,A,i} + P_{2,B,i})$
$z_{A,i}$
$z_{B,i}$
Alice:
$m_{A,i} = z_{A,i} / (r_1 * r_3) + r_5$
Bob:
$m_{B,i} = z_{B,i} / (r_1 * r_3) - r_5$

Fig. 9. Flow of Phase 2

Phase 3

Input: A has arrays $\mathbf{t_A}, \mathbf{m_A}$ and B has $\mathbf{t_B}, \mathbf{m_B}$

Output: A and B receive $w_{A,n}$, $w_{B,n}$ from server, where $w_{A,n} + w_{B,n} = v^u$.

Protocol: In this phase also, we use MP protocol. However, unlike earlier, here it is recursive in nature. We multiply all the elements in the arrays $\mathbf{t} + \mathbf{m}$. Since the splits (k_A, k_B) received from server are additive in nature, we add k_A to the existing $w_{A,i}$

3.1.6 Greater than Zero (GT0)

Input: A nonzero value v shared by A and B as v_A and v_B.

Output: A binary bit b, denoting whether $v > 0$, shared by A and B as two bits b_A and b_B, and $b = b_A + b_B \mod 2$.

Protocol:

1. A generates two shared random numbers r_1 and r_2, and sends $p_A = v_A / r_1 + r_2$ to S.
   B generates the same random r_1 and r_2, and sends $p_B = v_B / r_1 - r_2$ to S.
2. S adds p_A and p_B, and sets a bit b as 1 if the sum is greater than zero and as 0 if not. S additively splits b into two bits b'_A and b'_B, and sends them to A and B respectively.
3. If r_1 is positive, A and B set b_A and b_B as b'_A and b'_B respectively; otherwise, A sets $b_A = -b'_A$, but B sets $b_B = 1 - b'_B$.

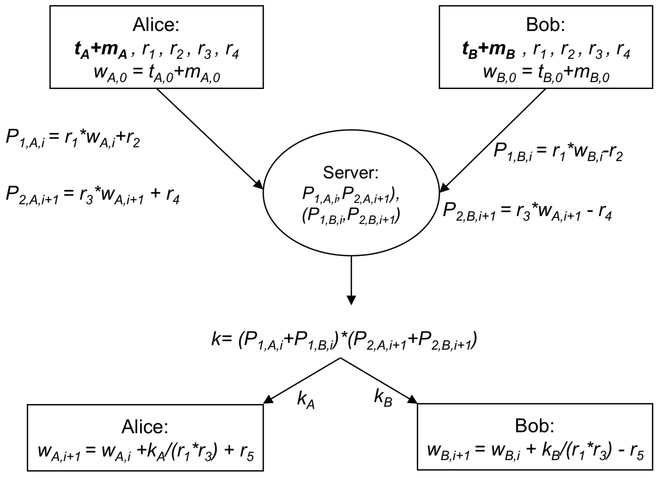
Alice:
t_A+m_A, r_1, r_2, r_3, r_4
$w_{A,0} = t_{A,0}+m_{A,0}$
Bob:
t_B+m_B, r_1, r_2, r_3, r_4
$w_{B,0} = t_{B,0}+m_{B,0}$
$P_{1,A,i} = r_1*w_{A,i}+r_2$
$P_{2,A,i+1} = r_3*w_{A,i+1} + r_4$
$P_{1,B,i} = r_1*w_{B,i}-r_2$
$P_{2,B,i+1} = r_3*w_{A,i+1} - r_4$
Server:
$P_{1,A,i},P_{2,A,i+1}),$
$(P_{1,B,i},P_{2,B,i+1})$
$k= (P_{1,A,i}+P_{1,B,i})*(P_{2,A,i+1}+P_{2,B,i+1})$
k_A
k_B
Alice:
$w_{A,i+1} = w_{A,i} +k_A/(r_1*r_3) + r_5$
Bob:
$w_{B,i+1} = w_{B,i} + k_B/(r_1*r_3) - r_5$

Fig. 10. Flow of Phase 3

Correctness: It is straightforward to verify this protocol works.

Security: From S's view, the inputs are hidden by r_1 and r_2, and the output bit b is perfectly hidden by the sign of r_1. A or B only sees an additive share of a bit, which will not leak any information of the output.

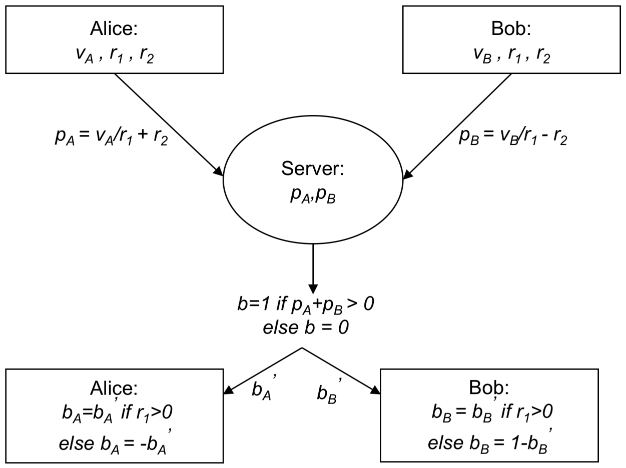

Fig. 11. Flow of the GT0 Protocol

3.1.7 EW0 protocol

In the high level protocols, A and B need to check whether some operand is zero as a branch condition in the code. For example, if a pivot is zero in the Gaussian elimination, they will call a row swap function. This protocol tells them whether the condition is true or false, without leaking the value if it is not zero, and assuming that in the setting up phase, A and B agreed a cryptographic hash function $H()$.

Input: A value v shared by A and B as v_A and v_B.

Output: A and B learn whether $v = 0$ or not.

Protocol:

1. A generates a local random number r_A and computes $x_A = r_A * v_A$, and sends it to B.

   B generates a local random number r_B and computes $x_B = r_B * v_B$, and sends it to A.

2. Please note that Alice is not aware of r_B and Bob is not aware of r_A.

3. A computes $y_A = x_B * r_A$.

   B computes $y_B = x_A * r_B$.

4. A computes $H(-y_A)$, and sends it to S.

   B computes $H(-y_B)$, and sends it to S.

5. S simply sends what it receives from A to B, and what it receives from B to A.

6. A receives the value $H(-y_B)$ from S. If $H(-y_B) = H(y_A)$, it learns $v = 0$; otherwise, it learns $v \neq 0$.

   B receives the value $H(-y_A)$ from S. If $H(-y_A) = H(y_B)$, it learns $v = 0$; otherwise, it learns $v \neq 0$.

Correctness: If $v = 0$, then $v_A = -v_B$, and thus $H(y_A) = H(-y_B)$; otherwise, $H(y_A) \neq H(-y_B)$. So, A and B learn the correct answer by checking the equivalence of the hashed values. Note that, in the computer representation with rounding issues, negligible values are usually treated as zeros, and the protocol can be modified to handle such cases smoothly as follows: Before performing the protocol, A and B truncate the most insignificant bits of v_A and v_B according to error tolerance setting. Security: Due to the one-way property of hash functions, each party cannot learn the true values by seeing the hashed values.

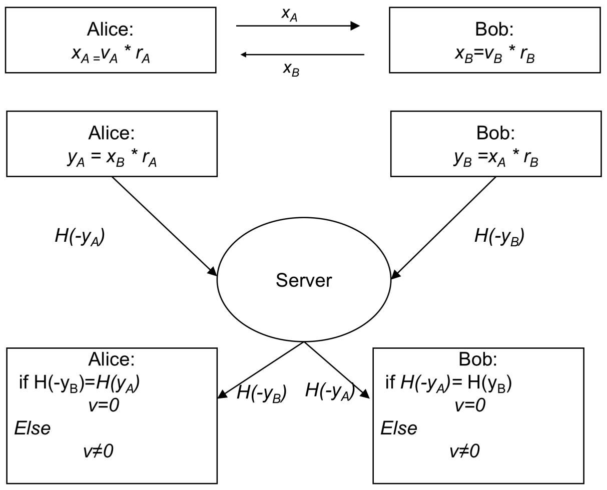

Fig. 12. Flow of the EW0 Protocol

3.1.8 Extension: Comparison under General Security Assumption

In the above GT0 protocol, we use non-cryptographic operations to compare a number with 0, and this is secure under our assumptions, and it may leak some information under the standard security assumptions. In this section, we provide a provable secure approach using the Garbled Circuit (GC) technique [14]. To make it complete, we provide a brief review in

Appendix A of how two parties use GC to securely evaluate a function without leaking their own private inputs. And in the following Secure-GT0 protocol, we use GC in a variant scheme, involving a server as the circuit evaluator to get rid of the expensive oblivious transfer protocols.

Input: A nonzero value v shared by A and B as v_A and v_B.

Output: A binary bit b, denoting whether $v > 0$, shared by A and B as two bits b_A and b_B, and $b = b_A + b_B \mod 2$.

Protocol:

1. A generates a random r_1, if $r_1 > 0$, A sets $x = v_A$, otherwise $x = -v_A$.
   B generates the same random r_1, if $r_1 > 0$, B sets $y = v_B$, otherwise $y = -v_B$.
2. A constructs a garbled circuit C, with x and y as inputs and a bit wire denoting whether $(x+y) > 0$ as output. A sends C to S and tells S which input wires representing x and which representing y.
   A shares with B the encodings chosen for the input wires corresponding to y, and shares the encodings chosen for the output wire with S.
3. A chooses the encodings representing x and sends them to S. B chooses the encodings representing y and sends them to S.
4. S receives the encodings for all the input wires, then evaluates C gate by gate, and gets the encoding of the output wire. S translates the output encoding into a binary bit b, and additively splits b into two bits b'_A and b'_B, and sends them to A and B respectively.
5. If r_1 is positive, A and B set b_A and b_B as b'_A and b'_B respectively; otherwise, A sets $b_A = b'_A$, but B sets $b_B = 1 - b'_B$.

3.2 Protocols to Additively Split the Input and Merge the Output

We assume the inputs to the co-design systems are matrices without loss of generalization, since single values and vectors are matrices with special dimensions. Each entry in the input matrices may be depending on (1) only A's design, i.e., A's private input, (2) only B's design, i.e., B's private input, (3) both A and B's design, and (4) only some constants or public information (like the laws of physics). We distinguish these 4 cases for any entry x, and present how A and B get the additive splitting form of it as x_A and x_B without leaking their private input.

(1) A computes x using its own input, generates a random r using the random seed known to B, and sets $x_A = x - r$.
B generates the same random r and sets $x_B = r$.

(2) A and B just switch the roles in case (1).

(3) Assume x depends on a set of A's private inputs I_A and a set of B's private inputs I_B. First, A and B additive split the input values in I_A and I_B as in (1) and (2), and then, they run the arithmetic protocols in 3.1 so that they learn x in its additively split form.

(4) A computes x only using these public information, generates a random r using the random seed known to B, and sets $x_A = x - r$.
B generates the same random r and sets $x_B = r$.

(Security) In case (1), (2) and (4), A and B do not communicate with others and thus do not learn any more information than what can be inferred from their own private input. In (3), they only communicate with the server S when performing the protocols in 3.1, which are already proved against information leaking.

Similarly, assuming the outputting result of the system is also a matrix. Each value y in it is shared by A and B as y_A and y_B after executing the protocols. Now we present the protocol enabling A and B to learn the final result without leaking to S.

1. A generates two shared random r_1 and r_2, and sends S $y_A + r_1$.

   B generates the same r_1 and r_2, and sends S $y_B + r_2$.
2. S simple sends what it receives from A to B, and what it receives from B to A.
3. A receives a value p_A from S and computes $y = p_A + y_A - r_2$.

   B receives a value p_B from S and computes $y = p_B + y_B - r_1$.

3.3 High Level Protocols

Based on the arithmetic building blocks, one can build higher level protocols to solve specific problems as stated in Section 2.

3.3.1 Vector Inner Product (VIP)

Input: A vector $\mathbf{v}$ of length n additively shared by A and B as $\mathbf{v}_A$ and $\mathbf{v}_B$; and a vector $\mathbf{u}$ of the same length additively shared by A and B as $\mathbf{u}_A$ and $\mathbf{u}_B$.

Output: w, additively shared by A and B as w_A and w_B, and $w = (\mathbf{v}, \mathbf{u})$.

Protocol:

1. A and B perform the MP protocol for all the n pairs of aligned values in the two vectors and thus get n intermediate results.
2. A and B perform the ASP protocol to sum these intermediate results to get w.

3.3.2 Check a Vector is Zero (CVZ)

Input: A vector $\mathbf{v}$ of length n additively shared by A and B as $\mathbf{v}_A$ and $\mathbf{v}_B$

Output: A and B know whether v is a zero vector.

Protocol:

1. A generates a random vector r_A of length n, B generates a random vector $\mathbf{r}_B$ of the same length, and they view them as an additive split of a vector $\mathbf{r}$.
2. A and B perform the VIP protocol with $\mathbf{r}$ and $\mathbf{v}$ as the inputs, result denoted as w.
3. A and B perform the EW0 protocol with w as the input. If $w = 0$, they know v is a zero vector, otherwise it is not.

It is with negligible probability that $w = 0$ while $\mathbf{v}$ is non-zero.

3.3.3 Matrix Multiplication (MM)

Input: A matrix M_1 shared by A and B as M_{1A} and M_{1B}; and a matrix M_2 shared by A and B as M_{2A} and M_{2B};

Output: M, additively shared by A and B as M_A and M_B, and $M = M_1M_2$.

Protocol:

For each entry m_{ij} in M, A and B perform a VIP protocol inputting the ith row of M_1 and the jth column of M_2, and fill the results into M_A and M_B respectively.

3.3.4 Matrix Full Rank Check (MFRC)

Input: A matrix M shared by A and B as M_A and M_B;

Output: A and B learn whether M is of full rank or not.

We use Gaussian elimination (row reduction) to check whether a matrix is of full rank. The elimination process involves three basic elementary operations, listed with its implementation on shared data as follows.

1. Multiplying a row with a non-zero number.

   This operation can be achieved by performing a set of MP protocols.
2. Multiplying a row with a non-zero number and adding the result to another row.

   This operation can be achieved by performing a set of MP protocols and an ASP protocol.
3. Interchanging two rows if a specific entry is 0.

   This operation can be achieved by interchanging the corresponding rows in both M_A and M_B, if A and B find the entry is 0 by performing the EW0 protocol.

During the elimination, A and B check whether the current processing row is a zero vector by the CVZ protocol. If they find a zero vector,then M is not of full rank.

3.3.5 Matrix Negative Definiteness Check (MNDC)

Let M be an $n \times n$ symmetric matrix, and let M_k denote the $k \times k$ submatrix formed by deleting the last $n-k$ rows and columns of M. The following two conditions are equivalent:(1) All the eigenvalues of M are negative, and (2) The determinants $(-1)^k \det(M_k) > 0$ for $1 \le k \le n$, i.e., $\det(M_1) < 0$, $\det(M_2) > 0$, $\det(M_3) < 0, \cdots, (-1)^n \det(M_n) = (-1)^n \det(M) > 0$. So, to check whether the eigenvalues of a shared matrix are all negative, A and B first compute $(-1)^k \det(M_k)$ for all M_k, and then compare these values with 0 with the GT0 protocol.

Now we discuss how to compute the determinant of any shared square matrix M of order n. If $n = 1$, $\det(M)$ is set to be the only number in M, otherwise, it can be computed by order reduction. Let m_{ij} denote the entry on the row number i and the column number j, for $i, j \in \{1, \cdots, n\}$, and set M_{ij} (called the cofactors) to be the determinant of the square matrix of order $(n-1)$ obtained from M by removing the row number i and the column number j multiplied by $(-1)^{i+j}$. Then $\det(M)$ can be computed by expanding the first row,

$$\det(M) = \sum_{j=1}^{j=n} m_{1j} M_{1j}$$

By this formula, $\det(M)$ can be eventually reduced to the sum of products of entries from M, so A and B can compute it by performing the *ASP* and *MP* protocols in the corresponding order.

4 Analysis of the Protocols

4.1 Security Analysis

In the security proof of the protocols, it is assumed that adding a large random number to a value hides it and multiplying a nonzero value by a large random number hides it. If the random number is generated from the infinite real range $(-\infty, \infty)$, it hides the value perfectly, since the sum or product of the value and such a random is uniformly distributed over the real range. However, in the machine world, depending on the number of bits used in representation, there is a range for the random numbers, say $[-b, b]$. Also there is a range for the values A and B are trying to hide, say $[-t, t]$. If S knows these two ranges, and gets a number near the bound $(b+t)$ as the sum of the value and a random, then it knows the value is near t. In order to prevent such leakage, A and B could either make b much larger than t so that it is not likely to generate randoms too close to the bound comparing with t, or let b vary when generating the random numbers and keep b secret from S. And we can see that the larger b is, the better it hides, while it costs more CPU time to compute.

4.2 Network Communication

During the running of the system, there is no need to build network connections between A and B, since our protocols do not require any interaction or data transfer between them. Therefore, this is a typical single-server-multiple-clients framework, and this model places light overhead on the client side, and the server is the only party responsible for listening to the communication socket. Moreover, this framework is easy to implement with standard network libraries.

All of those building block protocols only require a small constant round of client-server interactions, which is a desirable property for efficient network protocols. And in the higher level protocols, the building blocks can be executed in parallel to decrease the communication rounds. For example, in the VIP protocol, there is no dependence between the n times executions of the MP protocol, so A and B can send all the values that need to be sent in the n MP executions to S at once and get back the returned values from S in one interaction round.

5 Implementation and Results

Figure 13 represents the flow of the protocols used to measure the different properties of the collaborative system stated in Section 2.1. The implementation of the various protocols was carried out sequentially in the order shown.

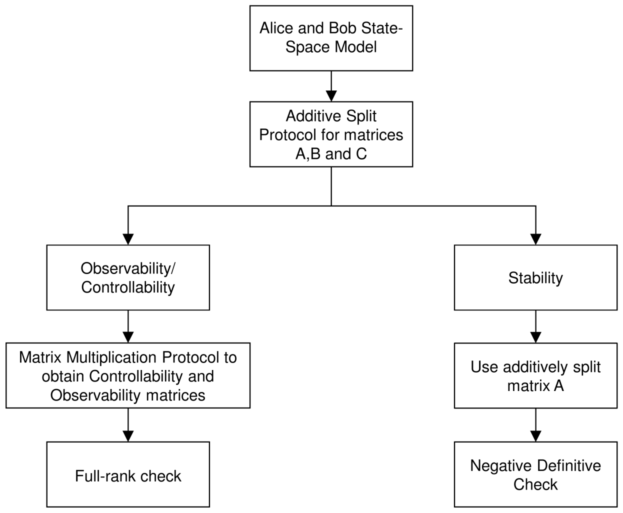
Alice and Bob State-Space Model
Additive Split Protocol for matrices A,B and C
Observability/ Controllability
Stability
Matrix Multiplication Protocol to obtain Controllability and Observability matrices
Use additively split matrix A
Full-rank check
Negative Definitive Check

Fig. 13. Implementation steps

We implemented the code for the protocols in C++ Language under MS Visual Studio. All the protocols are run in a single thread mode. Three separate projects were built to simulate the roles of Alice, Bob and the Server, and the server-client communication is implemented using the TCP protocol.

We first simulate the projects as three separate processes locally on a machine with Windows OS, CPU Intel i3 core 2.40 GHz, 6GB memory. The result is shown as follows.

As in [16], Alice's inputs are set as: $M = 4200$, $K_{fs} = 120$, $K_{rs} = 180$, $B_f = 25$, $B_r = 35$, $I = 40000$, $L_f = 55$, and $L_r = 65$; Bob's inputs are set as: $M_f = 125$, $M_r = 125$, $K_{ft} = 1100$, and $K_{rt} = 1100$.

1. The run time for additive splitting the matrices A, B and C and obtaining the observability and controllability matrices at Alice's side was 6054ms and at Bob's side was 138ms. The reason for such a difference is mainly because Bob's code runs after the connection is built between Alice and the server. Also Alice needs to do local computations for obtaining the additive splits as she has the knowledge of the structure of the collaborative system. Bob only has to input his private data following the protocols.
2. The run time for checking the rank of the controllability matrix was 28ms on both clients' ends. This collaborative system was controllable according to the output of the MFRC protocol.
3. The run time for checking the rank of the observability matrix was 45ms on both clients' ends. This system was observable according to the output of the MFRC protocol.
4. The run time for checking the negative definiteness of matrix A was 44ms on both clients' ends. This system was not stable according to the output of the MNDC protocol.

Table 1. Running time (in ms) of Alice in the two scenarios.

Protocols	Alice in Scenario 1	Alice in Scenario 2
Additive Split	11640	855214
Controllability	1184	142628
Observability	3017	307363
Negative Definiteness	1960	301605

To evaluate the performance over the Internet, we planted the server on one machine, and run Alice and Bob's code on another. Note that putting the two clients on the same machine or not does not make much difference to the performance, because the network communication is only between the server and a client, and our model does not assume Alice and Bob communicates directly. We measured the running time in two scenarios: (1) the server and clients are in the same campus, the server has fast cabled network, and the clients have fair wireless Internet connections, and (2) the server is physically a remote one in China, and the clients are in the US. The running time of Alice is described in the Table 1, and Bob is almost the same since most of the protocols are symmetric.

From the result we can see that the network delay is significant for the protocols, and this could be improved if we reduce the communication rounds of higher level protocols. As pointed out before, instead of calling the multiplication protocol n times in the VIP protocol, the clients can send to the server all the values to be sent in these n executions in one round. As a demo implementation, this optimization was not included in the experiment.

6 Closing Comments

The primary contributions of this paper within the field of engineering design are: a) formulation of the co-design problem in a form that allows secure privacy preserving simulation of system behavior, b) new protocols for the specific co-design scenarios discussed in the paper, c) demonstration of the practical viability of the protocols using a collaborative design example. One of the advantages of the protocols developed in this paper is that the cloud is not used as a repository. Data are exchanged with the cloud server for computations only. From the data exchanged, the server on its own (i.e., without colluding with any of the parties) cannot infer anything about the system properties or behavior. The paper also contributes to the cryptography literature. The computational building blocks presented in this paper and used in the protocols are highly likely to be useful for other scenarios and in other contexts beyond collaborative design.

We believe that this is a modest, but encouraging, initial step towards secure, privacy preserving model-based systems engineering (MBSE) and design. One of the assumptions in the co-design scenario chosen is that one of the parties knows the complete mathematical structure of the system, and only parameter values are private. There is significant opportunity to develop protocols for other co-design scenarios where this assumption may not hold. Finally, as highlighted in Section 1, our approach does not use encryption and other cryptographic primitives in order to reduce computation time. The use of modular arithmetic (for perfect hiding) will be investigated in the future work.

Acknowledgements

Portions of this work were supported by National Science Foundation Grants CPS-1329979, CNS-0915436, CMMI-1265622, Science and Technology Center CCF-0939370; by an NPRP grant from the Qatar National Research Fund; and by sponsors of the Center for Education and Research in Information Assurance and Security. The statements made herein are solely the responsibility of the authors.

References

[1] Horner, J., and Atwood, M. E., 2006. "Design rationale: the rationale and the barriers". In Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles, ACM, pp. 341–350.

[2] Bidarra, R., van den Berg, E., and Bronsvoort, W. F., 2002. "A collaborative feature modeling system". *Journal of Computing and Information Science in Engineering,* **2**(3), pp. 192–198.

[3] Red, E., French, D., Jensen, G., Walker, S. S., and Madsen, P., 2013. "Emerging design methods and tools in collaborative product development". *Journal of Computing and Information Science in Engineering,* **13**(3), p. 031001.

[4] Cera, C. D., Kim, T., Han, J., and Regli, W. C., 2004. "Role-based viewing envelopes for information protection in collaborative modeling". *Computer-Aided Design,* **36**(9), pp. 873–886.

[5] Wang, Y., Ajoku, P. N., Brustoloni, J. C., and Nnaji, B. O., 2006. "Intellectual property protection in collaborative design through lean information modeling and sharing". *Journal of computing and information science in engineering,* **6**(2), pp. 149–159.

[6] Peter Roehl, G., Raymond Kolonay, G., Rohinton Irani, G., Michael Sobolewski, G., Kevin Kao, G., and Michael Bailey, G., 2000. "A federated intelligent product environment".

[7] Howe, J., 2008. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business.* Crown Business.

[8] Terwiesch, C., and Xu, Y., 2008. "Innovation contests, open innovation, and multiagent problem solving". *Manage. Sci.,* **54**(9), pp. 1529–1543.

[9] Wu, D., Thames, J. L., Rosen, D. W., and Schaefer, D., 2012. "Towards a cloud-based design and manufacturing paradigm: Looking backward, looking forward". In Proceedings of the ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 2: 32nd Computers and Information in Engineering Conference, Parts A and B, ASME.

[10] Wang, H., and Zhang, H., 2013. "Designing by services: A new paradigm for collaborative product development". In *Cloud Manufacturing*, W. Li and J. Mehnen, eds., Springer Series in Advanced Manufacturing. Springer London, pp. 165–192.

[11] Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., and Molina, J., 2009. "Controlling data in the cloud: Outsourcing computation without outsourcing control". In Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW '09, ACM, pp. 85–90.

[12] Du, W., and Atallah, M. J., 2001. "Secure multi-party computation problems and their applications: a review and open problems". In Proceedings of the 2001 workshop on New security paradigms, ACM, pp. 13–22.

[13] Gentry, C., 2009. "A fully homomorphic encryption scheme". PhD thesis, Stanford University.

[14] Huang, Y., Evans, D., Katz, J., and Malka, L., 2011. "Faster secure two-party computation using garbled circuits.". In USENIX Security Symposium, Vol. 201.

[15] Wang, S., Nassar, M., Atallah, M. J., and Malluhi, Q. M., 2013. "Secure and private outsourcing of shape-based feature extraction". In ICICS, pp. 90–99.

[16] Klee, H., and Allen, R., 2011. *Simulation of Dynamic Systems With MATLAB and Simulink*. Taylor & Francis Group.

[17] Ogata, K. *Modern Control Engineering*. Instrumentation and controls series. Prentice Hall.

[18] Rabin, M. O., 2005. "How to exchange secrets with oblivious transfer". *IACR Cryptology ePrint Archive,* **2005**, p. 187.

Appendix A: Garbled Circuit Protocol

The basic idea to securely evaluate a function $f(x,y)$ by the GC protocol, where x and y are the private inputs from two parties, is to (1) construct the function f as a boolean circuit C, (2) encode x and y in their binary form as the input and the function value as the output, and (3) provide a method of computing such a circuit gate by gate in such a way that values obtained on all wires other than circuit-output wires are not revealed. To hide the values, for every wire in the circuit, two random encoding values are specified such that one value represents 0 and the other represents 1. Let w denote a wire, we use w_0 and w_1 to denote the two random encoding values chosen to represent 0 and 1 respectively.

Let g be a gate with incoming wires u and v and output wire w. Now we present how the gate g is garbled (i.e., to hide the true bit value of each wire by its random encoding value while still enabling the evaluation of the gate's output). This is accomplished by viewing the four possible input encoding values to the gate u_0, u_1, v_0, v_1 as encryption keys, and the two possible output encoding values w_0 and w_1 which are also keys, are encrypted under the appropriate keys from the incoming wires. For example, let g be an AND gate. Then, the key w_0 is encrypted under the pairs of keys (u_0,v_0), (u_1,v_0), (u_0,v_1), and the key w_1 is encrypted under the pair of keys (u_1,v_1). So, the gate g is garbled and represented as four ciphertexts $E_{u_0}(E_{v_0}(w_0)), E_{u_0}(E_{v_1}(w_0)), E_{u_1}(E_{v_0}(w_0)), E_{u_1}(E_{v_1}(w_1))$. Note that the four ciphertexts should be randomly permuted so that the position does not leak the meaning of the encoding values of wires. Now given the encoding values of the two incoming wires of a gate, one can, without knowing the meaning of input wires (i.e., whether it's 0 or 1) and the meaning of the gate (i.e., whether it's a logic *AND* or *OR*), try to decrypt the four ciphertexts, and only one will success and thus provide the proper encoding value of the output wire (we assume an "indicator of success" string of a pre-determined length is appended to each plaintext). By induction, given the appropriate encoding values of all the input wires to the whole circuit, one can compute the encodings values for the output wires gate by gate without knowing the meaning of the encodings.

In a garbled circuit protocol, one party P_1 (the circuit generator) prepares a circuit for computing f, chooses two encoding values randomly for each wire, and garbles all the gates as described. P_1 sends the garbled circuit to the second party P_2 (the circuit evaluator) to evaluate. P_2 obtains the encoding value for each input bit as follows: for those input wires corresponding to P_1's input, P_2 asks P_1 for the encoding values of them; and for the input wires corresponding to P_2's input, the two parties use oblivious transfer (OT) to enable P_2 to obtain the encoding values without leaking his input to P_1. An OT protocol [18] allows a sender holding encoding values w_0 and w_1, to transfer to a receiver holding a selection bit b in a way that the receiver learns nothing about w_{1-b}, and the sender learns nothing about b. With the encoding value of each input wire, P_2 then obliviously computes the output of the circuit without learning any intermediate values. For each output wire corresponding to P_2's output, P_1 sends both encoding values for 0 and 1 respectively to P_2 so that P_2 could interpret its own output; while for each wire corresponding to P_1's output, P_2 just sends the output encoding value to P_1, and P_1 could interpret since it knows the meaning of all the encodings.

List of Tables

1 Running time (in ms) of Alice in the two scenarios. 22

List of Figures

1 Co-design scenario 4
2 Collaborative half-car suspension system model 6
3 High-level overview of the protocol 8
4 Flow of the Addition and Subtraction Protocol (ASP) 10
5 Flow of the Multiplication Protocol 11
6 Flow of the Division Protocol 12
7 Flow of Logarithm Protocol 13
8 Flow of Phase 1 13
9 Flow of Phase 2 14
10 Flow of Phase 3 15
11 Flow of the GT0 Protocol 15
12 Flow of the EW0 Protocol 16
13 Implementation steps 21