Privacy Preserving Computation in Home Loans using the FRESCO Framework

Fook Mun Chan*, Quanqing Xu*, Hao Jian Seah†, Zhaohui Tang‡, Sye Loong Keoh†, Khin Mi Mi Aung*
*Data Storage Institute, A*STAR, Singapore
†University of Glasgow, UK
‡Singapore Institute of Technology, Singapore
Email: {chanfm, Xu_Quanqing, Mi_Mi_AUNG}@dsi.a-star.edu.sg,
woohaa3630@hotmail.com, Zhaohui.Tang@singaporetech.edu.sg, SyeLoong.Keoh@glasgow.ac.uk

***Abstract*—Secure Multiparty Computation (SMC) is a subfield of cryptography that allows multiple parties to compute jointly on a function without revealing their inputs to others. The technology is able to solve potential privacy issues that arises when a trusted third party is involved, like a server. This paper aims to evaluate implementations of Secure Multiparty Computation and its viability for practical use. The paper also seeks to understand and state the challenges and concepts of Secure Multiparty Computation through the construction of a home loan calculation application. Encryption over MPC is done within 2 to 2.5 Seconds. Up to 10K addition operations, MPC system performs very well and most applications will be sufficient within 10K additions.**

Keywords*—*Privacy; Secure Multiparty Computation; FRESCO

I. Introduction

Traditional methods of aggregating data for computing on a function relies on a trusted third party to perform the function. Consider the example of data analytics. Data analytics can only be done when a organization collects personal data about their users. This creates a huge privacy issue as companies can gain private insights of individuals based on such data, especially if this data is aggregated from multiple sources [1]. A simple example would be shopping habits of customers; a company can derive a person's health through the products that they buy. If a person constantly buys products that remove acne, data analytics can reveal that this person has acne, which is something that an individual might not want to reveal to a public entity.

Secure Multiparty Computation is a field of cryptography that explores joint computation of a function with inputs from different parties while keeping each party's inputs private. Secure Multiparty Computation can resolve these privacy issues as it generalizes the existence of a trusted third party into the security of cryptographic protocols. Research into specific fields like data mining [2] have been done with the same motivation, and shows the wide ranging use cases of the field. Secure Multiparty Computation is a subset of cryptography that has been not been used practically due to efficiency. However, recent developments in Secure Multiparty protocols have made it more efficient and more viable for practical implementation.

The aim of this paper is to create an application using a **Secure Multi Party Computation (MPC) framework** to compute home loan installments. This application will then be used to evaluate the MPC framework and determine the viability of MPC in practical usage. During the process of deciding to buy a home, buyers would calculate the required costs to determine if they are eligible and able to afford, which requires private inputs from different parties. The intention of this paper is to evaluate the use of the FRESCO (a FRamework for Efficient Secure COmputation) MPC framework in its current state and implementations of its Secure Multi Party Computation protocols. The focus is on usability and implementation of the framework. Since practical implementations of Secure Multiparty Computation are relatively new and undocumented, this paper seeks to implement a sample scenario to verify usability of current frameworks that implement Secure Multiparty Computation.

The rest of the paper is organized as follows. Section II describes the concepts and developments of Secure Multiparty Computation. Section III explores the context of home loans. Section IV defines the identified requirements that a Home Loan Calculation application should fulfill. Section V describes the system design. Section VI details the phases and description of the implementation of the Home Loan Calculation Application. Section VII discusses the experiments conducted to evaluate the implementation of FRESCO. Section VIII concludes this paper and potential future work.

II. Related Work

A. Secure Multiparty Computation

The concept of Secure Multiparty Computation was introduced by Yao in his paper introducing the classic millionaires' problem [3]. More specifically, Yao presents the problem as a generalized problem involving the use of multiple parties. Given a function $f(x_i \cdots x_n)$ and number of parties n, can function f be computed among the n participants among themselves such that each person P_i only knows its own input x_i and the output of function f? Yao's proposed solution for this problem in the paper is a secure two party protocol.

Yao's solution is based on party P_1 giving P_2 a list of possible values, with P_2 inputting his values into the list of possible values, upon which is returned to P_1 who is then able to securely evaluate a boolean function $f(x_1, x_2)$ by selecting the correct entry in the list of values to evaluate the function. While his solution is a two party secure computation, his generalization of the problem opened the idea of secure multiparty computation and contextualized it. There are two main secure multiparty computation approaches: *circuit garbling* and *secret sharing schemes*. Before explaining further on Secure Multiparty Computation concepts some terms must be defined. This section explains the terminologies that will be used in describing protocols for the rest of the paper.

1) Oblivious Transfer: In Oblivious Transfer, the sender sends a list of information to the receiver, while remaining unaware of what information that it has transferred. The above construction of solution is an example of Oblivious Transfer. Oblivious transfer is also used as a cryptographic primitive in many secure multiparty protocols.

An example of oblivious transfer is the 1 of 2 oblivious transfer. In this protocol, there is a sender Alice and receiver Bob. Bob desires a message from Alice, but does not wish Alice to know which information that Bob has requested [4]. Such a protocol can be implemented with any public key encryption. Generally, this protocol requires a few prerequisites. Alice as the sender has msg_0 and msg_1 messages that could potentially be the message that Bob desires. Bob has a bit b that corresponds to the message that he desires from Alice and does not want to let Alice knows which message he wants. The protocol can be implemented using any public key encryption schemes. This protocol has been generalized to a *1 out of n oblivious transfer* where the there can be more than two inputs [5].

2) Circuits: Logic Circuits are a model of computation for cryptography. A logic circuit is defined by their size depth i.e the length of their longest path. Logic circuits are also circuits whose operations are in Boolean [6]. They are often referred to Boolean Circuit in the cryptography literature.

B. Homomorphism

Gentry proposed a fully homomorphic scheme in his paper using lattices [7]. He defined *fully homormophic public encryption scheme* as a scheme that contains the functions: 1) f_{keygen} that generates the key, 2) $f_{encrypt}$ that encrypts a plaintext, 3) $f_{decrypt}$ that decrypts a cipher text, and 4) $f_{compute}$ that computes a circuit based on input ciphertext generated by $f_{encrypt}$ and outputs a ciphertext c that is result of the circuit. In addition to these functions, such a scheme should support any circuit. Gentry also describes different kinds of homomorphisms based on the lattice structure. These homomorphisms are *additive homomorphism* and *multiplicative homomorphism* [7].

1) Additive Homomorphism: Generally a scheme is *additively homomorphic* when plaintext values x and y satisfy the following condition:

$$f_{encrypt}(x) + f_{encrypt}(y) = f_{encrypt}(x + y)$$

The property implies that any addition of $cipher_i, \cdots, cipher_n$ ciphertexts using the same encryption scheme gives the same result when computing the plaintext.

2) Multiplicative Homomorphism: A scheme is *multiplicative homomorphic* when plaintext values x and y satisfy the following condition:

$$f_{encrypt}(x) \times f_{encrypt}(y) = f_{encrypt}(x \times y)$$

The property implies that any multiplication of $cipher_i, \cdots, cipher_n$ ciphertexts using the same encryption scheme gives the same result when computing the plaintext.

C. Yao's Garbling Circuit

Yao's influence for secure multiparty computation was extended further with his proposals to solve his original millionaire's problem. Known as Yao's Garbling Circuit, it relies on the use of circuits as a model of computation for computing a function. Using circuits, the idea is to encrypt the circuit to be computed, creating a "garbled" version of the circuit [8].

Yao's protocol starts with the *garbling/encryption* of the circuit. In this case we assume Alice and Bob, with Alice being the "garbler" and Bob being the "evaluator". Alice provides the circuit on which to compute on, which is garbled by her. Alice will send the garbled circuit and use the *oblivious transfer* primitive to send her garbled inputs to Bob. Bob then decrypts the circuit to obtain the encrypted outputs. Alice and Bob then communicate to reveal the final value of the output. Basically, the idea of the protocol is to provide a way to compute a function where values obtained on a circuit wire would not be revealed, with the exception of the output wire's value [9].

D. Shamir's Secret Sharing Scheme

Shamir [10] introduced a problem first formulated by Liu [11] as a background to his paper. Secret Sharing is a cryptographic primitive dealing with the problem of sharing a secret among n parties such that the secret can only be revealed upon combining t number of shares from the parties. Shamir's scheme is a *threshold scheme*, or a (k,n) *threshold scheme* [10]. Given the secret S divided into n parts, the following properties apply:

1) **Reconstructible**: Knowledge of k parts of the secret can easily reconstruct S,
2) **Secrecy**: Knowledge of $k - 1$ parts of the secret do not allow reconstruction of S; furthermore all permutations of S is possible at k-1 parts.

Since Shamir's scheme is based on interpolation of polynomials, every share of secret value u $[\![u]\!]$ is a point of $f(x)$. Given also a sharing of another secret v as $[\![v]\!]$ and n number of parties we observe that:

$$[\![u + v]\!]_i = [\![u_i]\!] + [\![v_i]\!]$$
$$u+v = Func_{dec}([\![u+v]\!]_i, [\![u+v]\!]_{i+1}, ...[\![u+v]\!]_n)$$

Adding different shares of different secrets creates a new share based on the sum of the secrets. When these secrets are shared with at least k parts, then we can compute the real value of the sum of the two secret values [12].

III. Background of Home Loans

This section details the background of home loans in Singapore. Described in this section includes the context and scenario needed to make the application work.

A. Overview of Home Loan Privacy

In Singapore, **Property Agents** are service people who help home buyers with the financial paperwork when purchasing a home [13][14]. These property agents also help consult potential buyers on the home that they wish to buy, which includes the financial aspect of affording the home. Typically, this presents *privacy problems*, as the calculation of the amount requires information that intrudes on the privacy of the home buyer (i.e., savings). In addition, when a property agent consults a buyer based on the financial aspects of the potential home purchase, the *lender* (i.e., banks) of any potential loan taken is not involved in the consultation. Finally, in Singapore, the social security system *CPF* can help pay for part of the home cost.

These entities (*buyer, bank, CPF*) are required for accurate computation of a home purchase, but they are not connected together; to do so would incur privacy loss on the part of the buyer, as the full calculation would reveal private information the user has in the three entities. The rest of this section shall elaborate on the details of the overview that is presented. Section III-B explains the overview in Singapore's context and section IV will show the high level overview of the application.

B. Context

In Singapore, up to 80% of the population stay in public housing built by the government, also known as **HDB flats** [15]. There are also different kinds of HDB flats, with different prices for each [16]. While there are also a sizable number of population who possess private housing, the application will explore the purchase of public housing flats as it is a more general case for a higher percentage of the population in Singapore. In addition, we can also generalize the scenario into a more global context.

The model for this application is using the Singapore housing context. This application uses Singapore's public housing payment model and conditions(HDB flats) for buying a flat to compute home loan installments. The rest of the section shall explain the concepts of the scenario in more detail.

1) Central Provident Fund: In Singapore, the Central Provident Fund (CPF) is a social security system that helps working Singapore Citizens and Permanent Residents (PR) to save enough for their retirement [17]. The scheme also provides the use of a citizen's/PR's funds for certain purposes like housing and health care.

CPF also allows use of funds for purchase of a house [18]. In particular, a buyer of a HDB home can pay part of the cost of the house using their funds held in CPF. The amount of which can be paid is dependent on various factors, most notably that it cannot exceed the amount that a user has in his account with CPF.

2) Total Debt Servicing Ratio And Monthly Debt Servicing Ratio: In Singapore, a condition for being able to take a loan from the bank is the *Total Debt Servicing Ratio* (TDSR) [19]. TDSR is a loan limit using a person's monthly income. For Singapore, the TDSR cap is 60% of a person's monthly income that a user can use to service his monthly debt repayments [20].

In HDB loans, the monthly debt servicing ratio (MSR) applies instead [21]. However, they are the same thing, just that the cap is different at 30% and only applies for HDB flats. We choose to generalize all *debt upper bound calculation* as **MSR** in the application.

3) Downpayment: The purchase of a HDB flat can be separated into two portions: the *downpayment* and the *loan*. This section will explain the downpayment portion of the scheme. When purchasing a HDB flat, a buyer can choose either a *HDB housing loan* or a *bank loan* [22]. Since they are both loans operating on similar principles, this section shall explain the **HDB loan** as a example for explanation.

HDB requires that if a bank loan is taken to pay for the purchase of a HDB flat then the buyer has to pay 20% of the purchase price as downpayment [22]. Of this 20%, at least 5% of the purchase price must be paid in cash, with the balance is payable using the buyer's *CPF* funds under the CPF scheme for public housing [18].

4) Home Loan: This section will explain the *loan* portion of the loan scheme that we are using. Home loans are *amoritizing* loans [23][24]. Amoritizing loans work by calculating interest on a annual basis. The interest is calculated by taking the outstanding amount owed and multiplying it by the interest. HDB loans are fixed rate loans pegged to the CPF interest rate. To calculate the monthly installment, we use the Equated Monthly Installment formula. The formula reads as follows:

$$A = P \cdot \frac{1-(1+r)^n}{(1+r)^n - 1}$$

Where A is the monthly installment, P the principal/outstanding amount, r the interest rate and n the repayment period in months.

IV. Requirements

This section will state the structure and purpose of the application created in this paper.

A. Problem Statement

Calculating the financial details of buying a new HDB flat is often a complicated process that requires private data of the buyer(i.e., savings, debt) from many different sources. These private data should ideally be secured from any other parties other than the buyer himself. However, current methods of calculation still require knowledge of the private values to allow actual calculation to happen. These problems are somewhat mitigated as the parties involved are segregated from one another, only using the *output* (i.e., Yes or No for checking if savings are enough) of each party to carry on the calculation. While this ensures the secrecy aspect, this can only be done when the actual purchase of home happens, a buyer would not be able to calculate the estimated costs securely as he needs to reveal information to a **property agent** for him to get consultation on his potential purchase.

B. The Solution

We aim to solve individual privacy by aggregating the three entities (*CPF, Bank, Buyer*) data for calculating estimated loan installment amount. This aggregation of private data will be done via the use of *Secure Multiparty Computation* techniques. The data we wish to protect are the buyer's monthly salary, CPF amount in **CPF**, **savings** and debt that are recorded in **banks**. Secure Multiparty Computation is a relatively new field of cryptography, and we analyze the potential uses of implementations of SMPC frameworks at its current state using this problem as a model to evaluate.

C. Parties

The three entities that were identified for the solution are: 1) *CPF*, 2) *Buyer*, and 3) *Bank*. Each party other than the buyer is required to provide some *private details* that they cannot share with any other party to calculate the monthly installment when purchasing a property selected by the buyer. In section IV-D, a high level description of what calculations need to be done is detailed, from which we can infer which values each party is require to provide for calculating a home loan's monthly installment.

In our solution, a *trusted third party* is not desired in computing the home loan; the only parties are the parties listed in this section, and they jointly compute the calculation together.

1) CPF: CPF only needs to provide one value: Amount Usable in CPF account. This value must remain secret, as the the amount that a buyer can use from his account must not be known to the bank. The value is needed as it is required for calculating the downpayment for a HDB flat.

2) Buyer: The buyer party is required to provide the following values:1) 30% of their monthly salary, 2) Repayment period in months, 3) Minimum amount of money for downpayment for chosen HDB flat, 4) Minimum amount of money required in CPF for the chosen HDB flat, 5) Maximum amount loanable for the chosen HDB flat. The only value here that needs to be secret is the buyer's *30% of monthly salary*. The remaining values can be public, as they are based on the buyer's choice of HDB flat. Those values are needed to calculate the monthly installment of a loan, and to verify the buyer's eligibility for a loan.

3) Bank: The bank is required to provide the following values: 1) Buyer's Existing Debt, 2) Buyer's Savings, and 3) Interest of loan. All the values from the bank except interest are required to be secret, as these are private details of the buyer. Interest is needed to calculate the loan's monthly installment.

D. Calculating a HDB Home Loan's Monthly Installment

Based on the context described in section III-B, there are several preconditions for getting a HDB home loan. They are as follows:

1) the Total Debt Servicing Ratio/Monthly Debt Servicing Ratio threshold,
2) Amount of cash the buyer has on hand to pay the downpayment,
3) the CPF funds usable to pay the downpayment.

A buyer has to ensure that he does not exceed the TDSR/MSR threshold, has enough money he has on hand and also enough money that he can use in his CPF account before he can be eligible to buy a HDB flat. Based on the context, we can detail the steps required to calculate a home loan.

1) Determine TDSR/MSR limit and see if loan is allowed to be acquired.
2) Determine if buyer's CPF funds and cash on hand is enough to pay the 20% downpayment.
3) Calculate the monthly installment using the *Equated Monthly Installment formula.*
4) Add the calculated installment value to existing debt and recheck TDSR.

V. System Design

Two experimental versions and a prototype of the home loan calculation application were created in this paper, in accordance to the requirements discussed in section III. This section details the different phases during the implementation and the decisions made on evaluating the technology used.

A. Overview

The system architecture is shown in Figure 1. The system implementation was conducted in phases. These phases were:

1) Evaluation of Protocols in FRESCO (as discussed in Section II)
2) Implementation of a simple interest calculation application prototype
3) Implementation of a amortizing loan calculation application prototype

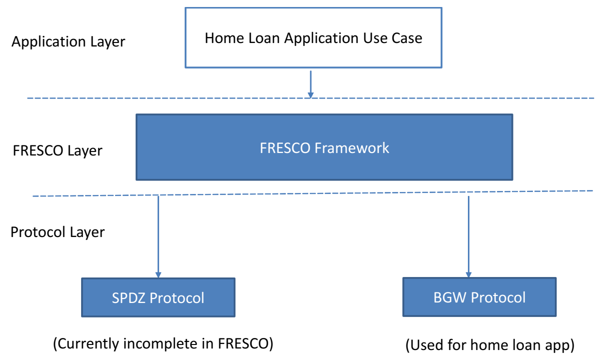


Fig. 1: System architecture

The implementation was conducted in the order shown above. Firstly, the frameworks *Sharemind* [25] and *FRESCO* were shortlisted and evaluated for use in implementing the requirements as discussed in Section IV. After choosing the framework, the protocols used in the framework were evaluated for use,with attempts to create simple prototypes and functions; this protocol evaluation will be discussed in section VI-A. After that, true implementation of the requirements as discussed in Section IV were created using the protocol. Two implementations were created, as *proof of concept* implementations because of technical reasons that will be discussed in Section VII.

B. Investigation of the FRESCO framework

FRESCO is a framework that is designed for users to easily write prototypes based on secure computation. It allows rapid and simple application and protocol suite development as well as a flexible design pattern with support for large and efficient computations [26]. FRESCO abstracts the idea of different protocol suites to create a plug and play framework. This is achieved by FRESCO's Protocol Producer/Consumer Pattern.

1) Usability: FRESCO is a framework that is easily extensible and flexible; users can define a protocol that they wish to use to evaluate a certain function. In that sense, protocols are *decoupled* from application development; developers just need to specify a function like addition that they wish to calculate without knowing about its specifics, which FRESCO calls the *Abstract Factory Pattern* [27]. FRESCO envisions that applications using this pattern can be run on multiple different protocol suites, using common operations to act as a abstraction from the protocols when developing.

FRESCO is a relatively new framework that has been around since 2015. Developed by Alexandra Institute's security lab [28], it is written in Java and licensed under the open source MIT license [26]. Currently it is in its first unstable version, version 0.1.0. It uses SCAPI as the underlying networking protocol for use in its application and currently has three protocol suites that are implemented.

2) Protocols: These protocol suites are:

- the **Dummy** protocol suite,
- the **BGW** Protocol suite,
- the **SPDZ** protocol suite.

The Dummy protocol suite has no security and is used as a measure for the basis overhead of FRESCO [29]. The BGW and SPDZ protocols will be explored in the later sections of this section.

The BGW protocol is a protocol used in the FRESCO framework. Proposed by Ben-or et al. [30]. It is a protocol which describes a way to implement secure multiparty computation for several logical operators. In particular, they defined circuits for addition and multiplication, and created a secret sharing scheme that would be secure in presence of an adversary [31]. The protocol is based on Shamir's Secret Sharing Scheme; in particular, BGW tweaks certain rules of Shamir's schemes so that they can compute operations using shares generated by the scheme. In general, secure computation in BGW consists of three steps:

1) **Input Sharing Stage**:
In the input sharing stage, each party P_i creates a share $[\![u_i]\!]$ using threshold $t+1$ where $t < n/2$ and distributes them among the parties.
2) **Computation Stage**: In this stage, parties jointly compute a function f using the values they hold. the function $f(x_i, ...x_n)$ will return a output $[\![out_i]\!]$. Each $[\![out_i]\!]$ is a sharing of the true value out. The function to be computed and their behavior depends on the formula to be calculated.
3) **Output Reconstruction Stage**: In this stage, parties collude and communicate to reconstruct the output out by using shares of $[\![out_i]\!]$ from all parties $P_0,,,,P_n$ parties. If only one party is required to know the output, all parties send shares to the party that is only allowed to know the output.

SPDZ is a protocol that was developed in 2012. It is implemented in the FRESCO framework. The protocol differs from the BGW scheme in several ways, notably in the use of Message Authentication Code (MAC) for authenticating shares, and the use of a somewhat homomorphic scheme(SHE) during preparation of values to be computed in the protocol [32]. However SPDZ is also a *secret sharing* scheme, like BGW. Computing operations like addition and multiplication is different as compared to BGW with some novel notable concepts. In particular SPDZ consists of a two phase protocol: 1) *Preprocessing Phase* and 2) *Online Phase*.

3) The Contrasting Design Philosophies: Sharemind is a commercial application design for commercial usage. In contrast, FRESCO is open source, which allows anyone who wishes to use or contribute to the framework instant access. Since the system is time bounded, FRESCO's instant usability clearly is better suited. In addition, Sharemind's design is as a framework that provide a full suite of functions for secure multiparty computation; this means that any application written with Sharemind must be in Sharemind's context. FRESCO, however, envisions itself as a *plug and play component* in a larger application. In this case, FRESCO is better suited for the system's purpose, as we also seek to *evaluate the general use* of secure multi party computation frameworks.

VI. System Implementation

A. Evaluation of Protocols in FRESCO

FRESCO is a *platform* for secure multiparty computation protocol implementations. In this section, the protocols implemented in FRESCO 0.1.0 are evaluated and one protocol will be chosen for the implementation of the home loan calculation application as described in Section IV. There are three secure multiparty computation protocols that are implemented in FRESCO. They are: Dummy, BGW protocol and SPDZ protocol. The **dummy** protocol is a protocol that is used for measuring FRESCO's overhead. It provides *zero security* and thus not usable for the actual home loan calculation application for the system.

A major problem in FRESCO is that it **does not allow decimals** in the framework. This applies to both SPDZ and BGW. The *SPDZ* and *BGW* protocols were evaluated against each other to determine which of the two protocols were to be used in implementing the requirements as described in Section IV. The implementation for both were studied by creating a prototype application that does simple addition and multiplication. These efforts are detailed in this section.

1) SPDZ: A prototype of SPDZ was attempted to evaluate the protocol for use in the system. During the attempt to build the prototype, flaws in the implementation of SPDZ was discovered. These flaws are:

- The preprocessing phase was not implemented fully in SPDZ.
- The utility class cannot parse specified SPDZ options properly.

A working prototype of SPDZ was attempted but not completed, as the actual implementation of SPDZ in FRESCO is incomplete; while FRESCO has a method that lets a *trusted* *party* to generate the preprocessing requirements, using this third party would violate privacy as we do not want the values from each party to be known to any other party other then the party inputting the values itself.

A prototype was created successfully for BGW that does simple addition and multiplication. However, just like SPDZ, flaws in the implementation of the protocol was discovered. These flaws are:

- Negative values are not supported in BGW due to implementation bugs.
- The framework's utility class is unable to parse user specified BGW options due to bugs.
- If a computation returns a negative value it returns the $modulus - (negativevalue)$.

These bugs and problems will be further explained later in section VI-C1.

2) Choice of Protocol: **BGW** was chosen as the protocol to use in FRESCO as it is the only protocol that does not require any other party then the ones identified in Section IV. In addition, the SPDZ implementation in FRESCO is still a *work in progress*; examination of the implementation shows that even though a method of doing the *preprocessing phase* is implemented, it is not a proper implementation but a place-holder for a future full implementation of the preprocessing phase.

B. Actual Implementation of Application

There are two home loan calculation applications that were produced for this system. One is based on a **simple interest** scheme, and the other is based on a **amortizing interest** scheme. This section presents the implementation of the home loan application. Firstly, the definitions of each component required for the home loan application will be first described as follow. This will be followed by the actual explanation of the application workflow in section VI-C, which lists the components of the application in the order that it happens.

Firstly, to simplify the explanation of the implementation of the application, we shall define some terms. We define the BGW protocol stages as:

- the Input Sharing Stage as $Stage_{input}$
- the Computation Stage as $Stage_{compute}$
- the Output Reconstruction Stage as $Stage_{output}$

Here, we define the terms relevant to the parties involved as detailed in Section IV. We define the inputs from the party **buyer** as follows:

- 30% of monthly salary as $salary$
- Repayment Period as $Paymentperiod$
- Chosen Flat's minimum cash required as $requiredcash$
- Chosen Flat's minimum CPF amount needed as $CPFNeeded$
- Chosen Flat's maximum loanable as $MaxLoanable$

For the party **bank**, we define the inputs:

- Buyer's Existing Debt as $Debt$
- Buyer's Savings as $Savings$
- Interest rate of Loan as $interest$

For the party **CPF** we define its input *amount usable in cpf account* as CPF_{Usable}.

C. Application Workflow

Two versions of the home loan calculation application were created for the system. The only difference between the two is the calculation of the monthly installment; namely **simple interest** and **amortizing interest** schemes. Apart from the formula used to calculate the interest, the two versions are the same. This section will detail the **generic application work flow for both versions** and detail the differences where it happens.

1) Assumptions of Application: This home loan calculation application will work under a few assumptions based on the bugs that were identified in FRESCO's implementation of BGW. They are:

- $savings \geq requiredcash$
- if the result of any computation is more than 60,000,000,000, it is a negative value. This value is defined as $bound$

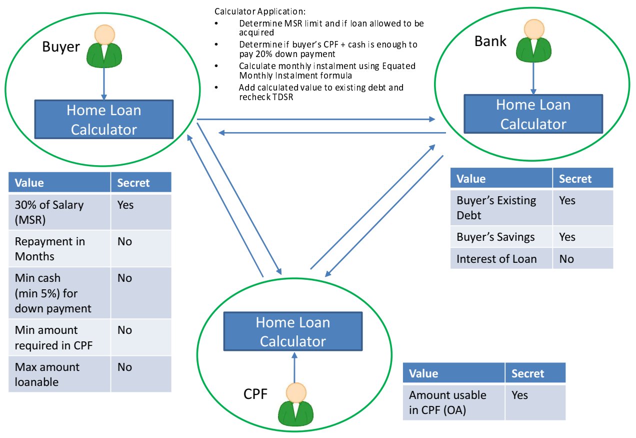
We assume that when a buyer wants to calculate the home loan cost using the application, he should know that he has enough money in his savings to calculate his costs required.

To address the bug of BGW returning the modulus, we assume any number above a certain threshold is a negative value. We take the number 60,000,000,000 for our threshold as it is sufficiently high enough such that it is improbable that the value returned by $modulus - 60,000,000,000$ is a realistic number.

2) Gathering Input: The application will first require users to identify themselves. This is done through a command line interface, requiring users to input a number that identifies the party that the user is. Each party runs on a different address port based on their party identification ID. The network address for the three different parties in the application is set in the code; all parties are required to know the address of all other parties so that they can share their inputs with each other for the *Computation Stage*.

After specifying the party, the user of the application will be prompted to enter values based on the party that they have identified as. For example, a user of an application who identifies himself as the *CPF* party has to input the CPF_{usable} amount. Although discussed in Section IV some parties' values need not be secure, the application will still use the networking implementation in FRESCO to simplify the implementation. At the end of this part, each party should have their values usable in this stage.

3) Calculating the Home Loan: After gathering all inputs required, we use FRESCO's networking implementation to invoke $Stage_{input}$ for secret sharing the inputs among all parties. Using these inputs, we begin the actual secure computation to determine the home loans, as shown in Figure 2.


Calculator Application:
• Determine MSR limit and if loan allowed to be acquired
• Determine if buyer's CPF + cash is enough to pay 20% down payment
• Calculate monthly instalment using Equated Monthly Instalment formula
• Add calculated value to existing debt and recheck TDSR
Buyer
Home Loan Calculator
Bank
Home Loan Calculator
Home Loan Calculator
CPF

Value	Secret
30% of Salary (MSR)	Yes
Repayment in Months	No
Min cash (min 5%) for down payment	No
Min amount required in CPF	No
Max amount loanable	No

Value	Secret
Buyer's Existing Debt	Yes
Buyer's Savings	Yes
Interest of Loan	No

Value	Secret
Amount usable in CPF (OA)	Yes

Fig. 2: FRESCO-based Home Loan Calculator Application

The payment for a loan can be divided into the 20% downpayment and 80% loanable amount. In the 20% downpayment, *at least 5%* is required to be paid by cash, with the rest being payable by a buyer's CPF account. This 5% is a lower bound, and can be higher if the buyer wishes so.

However, we wish to check if the CPF account has enough money to be able to afford the house. We do so by the following formula $f_{checkCPF}$:

$$CPF_{usable} + (savings - downpayment) - CPF_{needed}$$

We also wish to determine the monthly installment for a *amortizing home loan*. We do so by the following formula $f_{amortize}$:

$$A = P \cdot \frac{r(1+r)^n}{(1+r)^n - 1}$$

where $r = interest/12/100$ and $n = Paymentperiod$.

This formula requires division and exponential functions, both of which are not defined in the BGW protocol nor implemented in FRESCO. We solve this by doing the calculation using values that are opened and known to the public, where traditional Java has functions for division and exponentials. We thus have to reveal the values $interest$, $paymentperiod$ and the result of $f_{checkCPF}$.

Alternatively, in another version of the application we use a *simple interest* loan calculation. This formula f_{simple} is

$$\frac{P \times r}{n}$$

Finally, we also wish to compute the TDSR of a buyer. We do so by the formula f_{tdsr}:

$$salary - debt - monthlyinstallment$$

Algorithm 1 shows how the functions that are securely computed after the parties provide their respective inputs. All parties have shares of every input (line 4).

Algorithm 1 High level description of Application Secure Computation Implementation

```
1: All parties input their values and identities into the application.
2: All parties invoke $Stage_{input}$ to secret share their inputs among all parties.
3: **Calculating using BGW** - the steps in this section is done in a secure way.
4: We construct a circuit $C_{computecpf}$ to compute $f_{checkCPF}$ by setting $CPF_{excess} \leftarrow CPF_{usable} + (savings - downpayment) - CPF_{needed}$.
5: We then construct a circuit $C_{revealCPF}$ to reveal the secret value $CPF_{excess}$.
6: We compute the amount loanable by constructing a circuit $C_{loanable}$ computing $loanable \leftarrow MaxLoanable - CPF_{excess}$.
7: We then construct a circuit $C_{revealloanable}$ to reveal the secret value $loanable$.
8: Finally, we wish to compute $f_{tdsr}$. Since installment is not known until the full computation is computed, first construct circuit $C_{loanlimit}$ computing $loanlimit \leftarrow salary - debt$.
9: Construct circuit $C_{revealllimit}$ that reveals the secret value $loanlimit$.
10: Construct circuit $C_{reveal}$ that reveals secret values $paymentperiod, interest$
11: Glue the circuits together in the order $C_{computecpf}$, $C_{revealCPF}$, $C_{loanable}$, $C_{revealloanable}$, $C_{loanlimit}$, $C_{revealllimit}$ and evaluate them.
```

Because of operation limitations of the protocol, we have to compute $f_{amortize}$ and f_{tdsr} using secret values made public. This computation will happen after Algorithm 1. Algorithm 2 details how the application considers if a buyer is eligible to buy a flat and the calculations required.

Algorithm 2 High level description of Application Non Secure Implementation

```
**Require:** Algorithm 1 was complete prior to this algorithm.
**Ensure:** Output that reveals the installment of the loan if buyer is eligible.
1: After Algorithm 1, we have revealed values $CPF_{excess}$, $loanable$, $loanlimit$, $paymentperiod$ and $interest$.
2: Compute $f_{amoritize}$ using $paymentperiod, loanable$ and $interest$ by $installment \leftarrow loanable\frac{interest(1+interest)^{paymentperiod}}{(1+interest)^{paymentperiod}-1}$.
3: Compute TDSR $TDSR \leftarrow loanlimit - installment$.
4: Check if $downpayment > bound$. If so this means that the downpayment is insufficient. Return an error message informing user that CPF is insufficient and exit.
5: Check if $TDSR > bound$ If returns true then this means that TDSR is a negative value, which means the user cannot get another loan. Return an error message that TDSR has been exceeded.
6: If none of the conditions evaluate to true, return the installment value $installment$ to the user.
```

The application algorithm starts with Algorithm 1 and then Algorithm 2. These algorithms describe the **amortizing**

interest version of the home loan application. For the **simple interest** version, the difference is adding a step to Algorithm 1 to calculate $loanable \times interest$ and in step 2 of Algorithm 2; replace the formula with f_{simple}.

Figure 3 shows the activity diagram of the application. Computation of the loan is done not in FRESCO's MPC framework, the reasons of which are explained in Section VI-C1.

VII. Performance Evaluation

This section discusses the evaluation of the feasibility of using the FRESCO framework to build a full application. We conduct an **Efficiency Evaluation** to decide if the computational overhead of FRESCO is suitable for use in a full fledge application. The experiments were conducted on a machine with the following specs: Windows 10 Home with Intel Core i7-4720HQ Processor, 16GB RAM and JVM heap size 2GB.

A. Efficiency Evaluation

A evaluation was done on the efficiency of the **framework**. This was done by two experiments, **measuring the time for encrypting values for secret sharing** and **measuring the time taken for computing an operation**.

1) Measuring Time taken for Encrypting Values: In this experiment, FRESCO's secret sharing implementation is evaluated. In particular, the time taken to encrypt a value for secret sharing is evaluated. In addition, we wish to evaluate if high values could be efficiently transformed to its encrypted secret shared form. The experiment was conducted by coding a custom test application that only encrypts the values that are retrieved from the various parties, with time measurement using Java's *System.currentTimeMillis()* function. A variable $scale$ was used to scale up the values, in order to determine if higher values would yield different timings of the function.

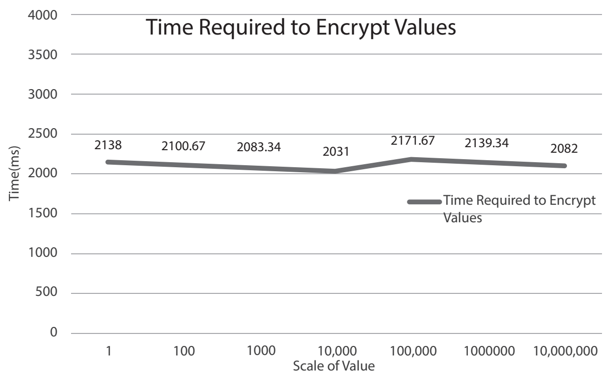


Fig. 4: Results from Measuring Time Taken for Encrypting the Values

The results were then plotted out in a graph in Figure 4. Figure 4 shows that the time taken to encrypt the data remains a constant regardless of how the values are scaled; This implies that the secret sharing implementation of BGW by FRESCO is a constant time implementation, which is quite efficient as the time taken is also quite low.

2) Measuring Time taken for Computing Addition: In this experiment, the potential overhead of operations in FRESCO's BGW implementation was evaluated. For this experiment, the addition operation was selected for evaluation. The experiment also evaluates and verifies if more addition operations would lead to cause an exponential overhead during computation.

The experiment was conducted with a custom application that does addition according to Algorithm 3. In addition, another application was created to compute simple addition using the same idea as *Step 3* of algorithm 3 but in a non secure traditional way in Java for comparison and context to the time of the application. A scaling factor of 10 was used to increase the number of additions for the experimentation.

Algorithm 3 Addition Method for Experimentation

```
Require: Number of times addition is to be done defined as
    $rounds$.
1: Invoke $Stage_{input}$ to get a value $val$ to compute addition.
2: Start Timer.
3: Create a circuit $C$ to based on the number of $rounds$
   specified.
4: This is done using a for loop and assigning $val = val+val$
   for each $round$.
5: Execute circuit $C$.
6: Stop timer and print out time taken.
```

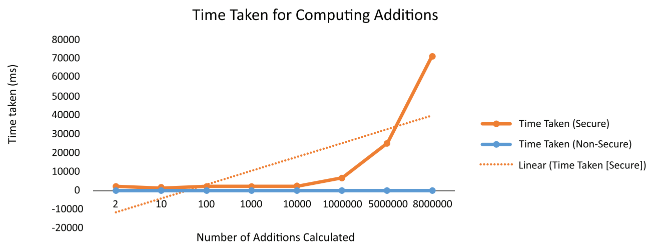


Fig. 5: Results from Measuring Time Taken for Computing Addition

The result of the experiment was plotted onto a graph shown in Figure 5. The results show that having more additions seem to be a exponential in time. Compared to traditional addition in Java, the time overhead cost of addition in BGW is noticeably higher, but still efficient. Efficiency only peaks at 1 million addition operations, which is a upper bound that is hard to reach for a conventional application

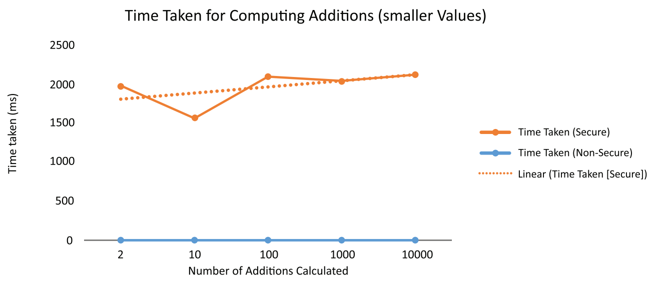


Fig. 6: Results from Measuring Time Taken for Computing Addition (No of Additions 2 - 1000)

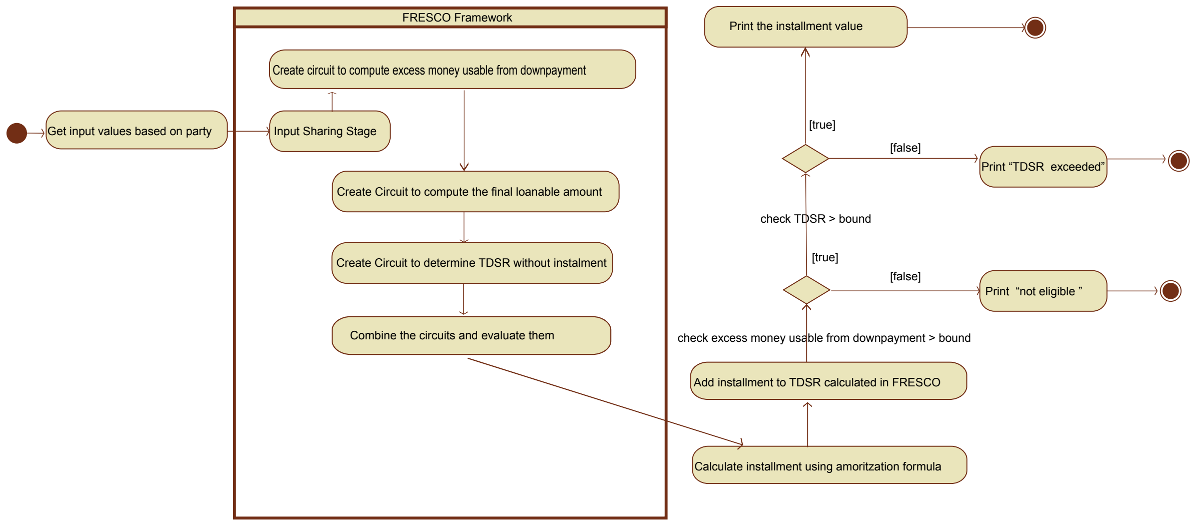

FRESCO Framework
Get input values based on party
Input Sharing Stage
Create circuit to compute excess money usable from downpayment
Create Circuit to compute the final loanable amount
Create Circuit to determine TDSR without instalment
Combine the circuits and evaluate them
Calculate installment using amoritzation formula
Add installment to TDSR calculated in FRESCO
check excess money usable from downpayment > bound
[true]
[false]
Print "not eligible "
check TDSR > bound
[true]
[false]
Print "TDSR exceeded"
Print the installment value

Fig. 3: Activity Diagram of the Home Loan Computation Application

While at first glance more addition operations seem like having exponential time overhead, examination of the smaller values reveals that the the time overhead may be linear instead. Since the experiment uses a scaling factor of 10, each time the number of additions $round$ is scaled the values are scaled exponentially. This might have affected the findings, and more values could be used to better determine the time overhead of the additions. However, the Figure 6 also shows a linear best fit line for the values. In addition with the observation of the smaller values, we can conclude that the time overhead of addition is a steep linear function.

VIII. Conclusion and Future Work

The aim of this paper was to implement a MPC scenario and use it to evaluate the practicality of existing MPC frameworks. Encryption over MPC is done within 2 to 2.5 Seconds. Up to 10,000 addition operations, MPC system performs very well and most applications will be sufficient within 10,000 additions. We believe that this aim was sufficiently achieved, but more work can be done to evaluate more frameworks and their implementations. For example, a implementation of SPDZ has been released on github by the institution that first proposed it [33]. Secure Multiparty Computation is a field which has indirect links to other sections of cryptography. We believe that elements of MPC, like secret sharing, can be used in tandem with other cryptographic techniques to enhance them, like key management schemes. In addition, MPC is based on homomorphic properties. The recent advances in Fully Homomorphic Encryption (FHE) is also a potential avenue of further study.

References

[1] A. Katal, M. Wazid, and R. Goudar, "Big data: issues, challenges, tools and good practices," in *Contemporary Computing (IC3), 2013 Sixth International Conference on*. IEEE, 2013, pp. 404–409.

[2] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 439–450.

[3] A. C. Yao, "Protocols for secure computations," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 1982, pp. 160–164.

[4] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, vol. 28, no. 6, pp. 637–647, 1985.

[5] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. ACM, 1999, pp. 245–254.

[6] J. E. Savage, *Models of computation*. Addison-Wesley Reading, MA, 1998, vol. 136.

[7] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *STOC*, vol. 9, no. 2009, 2009, pp. 169–178.

[8] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

[9] Y. Lindell and B. Pinkas, "A proof of security of yaos protocol for two-party computation," 2006.

[10] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[11] C. L. Liu, "Introduction to combinatorial mathematics," 1968.

[12] A. Beimel, "Secret-sharing schemes: a survey," in *International Conference on Coding and Cryptology*. Springer, 2011, pp. 11–46.

[13] "Our services - era realty network singapore," website, http://www.era.com.sg/about-us/our-services/ Accessed 2017-3-18.

[14] "Propnex - our property services and tool," website, https://www.propnex.com/servicesandtools Accessed 2017-3-18.

[15] H. . D. Board, "Key Statistics: Annual HDB Report," http://www10.hdb.gov.sg/eBook/AR2016/key-statistics.html, 2016.

[16] "Types of flats — hdb infoweb," website, http://www.hdb.gov.sg/cs/infoweb/residential/buying-a-flat/new/types-of-flats&rendermode=preview/ Accessed 2017-3-18.

[17] "Cpf overview," website, https://www.cpf.gov.sg/Members/AboutUs/about-us-info/cpf-overview Accessed 2017-3-18.

[18] "Public housing scheme," website, https://www.cpf.gov.sg/Members/Schemes/schemes/housing/public-housing-scheme Accessed 2017-3-18.

[19] M. A. of Singapore, "Mas introduces debt servicing framework for property loans," Jun 2013, http://www.mas.gov.sg/news-and-publications/media-releases/2013/mas-introduces-debt-servicing-framework-for-property-loans.aspx Accessed 2017-3-19.

[20] "Total debt servicing ratio (tdsr) simplified -stproperty," website, http://housingloansg.com/hl/resources/housing-loan-guide/tdsr-and-msr Accessed 2017-3-18.

[21] "Still confused by the tdsr framework? you wont be after reading

this -moneysmart.sg," website, http://housingloansg.com/hl/resources/housing-loan-guide/tdsr-and-msr Accessed 2017-3-18.

[22] "Costs and fees — hdb infoweb," website, http://www.hdb.gov.sg/cs/infoweb/residential/buying-a-flat/new/finance/costs-and-fees Accessed 2017-3-18.

[23] "How amortization works - examples and explanation," website, https://www.thebalance.com/how-amortization-works-315522 Accessed 2017-3-18.

[24] "The first-timers guide to the hdb loan," website, http://bennyblogger.com/first-timers-guide-to-hdb-loan/ Accessed 2017-3-18.

[25] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[26] "Introduction - fresco 0.2.0 documentation," website, http://fresco.readthedocs.io/en/latest/intro.html Accessed 2017-3-14.

[27] "The design of fresco - fresco 0.2.0 documentation," website, http://fresco.readthedocs.io/en/latest/design.html Accessed 2017-3-14.

[28] "Github- aicis/fresco: Framework for efficient and secure computation," website, https://github.com/aicis/fresco Accessed 2017-3-14.

[29] "Protocol suites- fresco 0.2.0 documentation," website, http://fresco.readthedocs.io/en/latest/protocol_suites.html Accessed 2017-3-14.

[30] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 1–10.

[31] G. Asharov and Y. Lindell, "A full proof of the bgw protocol for perfectly secure multiparty computation," *Journal of Cryptology*, vol. 30, no. 1, pp. 58–151, 2017.

[32] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 643–662.

[33] "Github-bristolcrypto/spdz-2 : Multiparty computation with spdz online phase and mascot offline phase," website, https://github.com/bristolcrypto/SPDZ-2/ Accessed 2017-3-18.