TRUSTONIC | Mimer

Mimer Trust: Efficient and Secure Data Sharing for Trusted Execution Environment in Embedded Systems [White Paper]

Abstract.

A Trusted Execution Environment (TEE) is a widely applied mechanism in embedded systems to protect sensitive data against adversaries. However, its clear isolation of the secured world from the normal world also restricts data sharing between the trusted and standard applications, which is desired in many modern embedded systems. Mimer Trust is a solution based on Mimer SQL and Trustonic's Secure Tablestore running inside the TEE to achieve fine-grained access control and efficient data sharing between the two worlds. The solution builds up relational storage for the secure data in the TEE, which are stored initially as simple binary objects. We provide a high-level API to manipulate the secure data and introduce SQL extensions to allow normal applications to access the Secure Tablestore with ordinary SQL statements and rich data types. Since all data access is made through the database server, privileges can be explicitly granted to normal applications, achieving fine-grained access control over the secure data. Encryption of data and data communication is used to protect the information. Mimer Trust provides a two-layered security mechanism with strong protection for sensitive data and flexible access policies suitable for various application scenarios.

1. Introduction

With the rise of intelligent and connected devices, there is a significant need to take security threats seriously. Smart devices may handle personal data that must be protected against theft or tampering. Embedded devices may have security secrets or service logs that would prove invaluable to an attacker. For example, in vehicles, the integrity of sensor data is critical to ensure that a speedometer is accurate and that "black box" data can be "trusted" and used in court or to diagnose or remediate a problem after an accident.

Modern embedded processors support a security system called a Trusted Execution Environment (TEE), also known as the Secure World. This has been widely adopted as a powerful mechanism to protect both code and data stored in mobile and embedded systems. In a TEE, Trusted Applications (TA) run in an isolated secure operating system with isolated memory and protected storage. Trusted Applications typically offer services to "regular" applications running in the normal operating system. These are referred to as Client Applications (CA). Client and Trusted Applications communicate through protected channels provided by the lowest levels of the device firmware, providing a post-box protocol to exchange memory buffers and ensure that the TEE cannot be subverted by malware from the Regular Execution Environment (REE), also referred to as the Normal World.

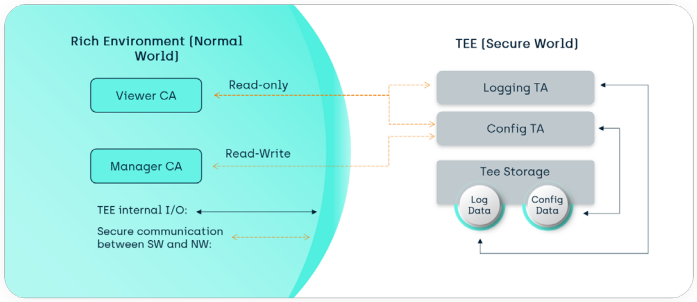


Fig. 1. Applications within a data box

The TEE provides strong protection for sensitive data owned by a trusted application. The trusted application, in turn, controls what access to the data is permitted by the client application. This CA-TA interaction is governed by the Global Platform Client API[1]. This low-level API is described in terms of memory buffers but is not well suited for direct use by application developers.

Motivating Example

To illustrate this, let us consider a "data box" deployed in a truck for recording the truck status and driver activities. In this system (Fig 1), two trusted applications run in the TEE, while two client applications run in the REE. A trusted application, Logging, keeps track of log data, while the other trusted application, Config, manages the configurations. To minimise data exposure while supporting rich functionalities, the desired feature is to grant client applications with minimal and different rights for accessing the data through the trusted applications. For instance, the Viewer client should only read the log and configurations, while the Manager client is permitted to modify the configuration.

Using vanilla Global Platform TA APIs, developers need to explicitly program the trusted Config application trusted Config application to distinguish Viewer and Manager to deny certain requests. As it is desirable for the trusted application to only respect requests from the appropriate client application, the implementation of the trusted application may need to be updated if a new client application is added.

The second enhancement to the current mechanism lies in high-level support for rich data types. Global Platform APIs are simple memory buffers. To express more complex data, developers need to explicitly implement the data structures and ensure they are consistent in both the trusted application and all its communicating client applications. Debugging of data-related errors is not trivial due to the ad-hoc interpretation of the structures. This does entail not only development efforts but also increases the risk of introducing bugs.

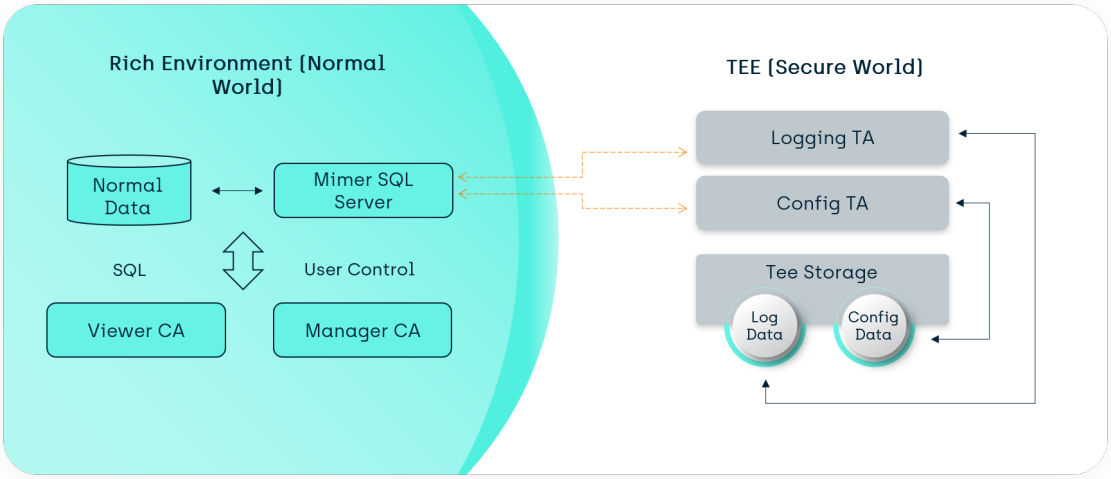


Fig.2. Databox with Mimer Trust

The Mimer Trust Solution

Mimer SQL is a commercial relational DBMS designed to support various systems, including the most popular embedded ARM platforms. Together with Trustonic, a state-of-practice TEE provider for embedded systems, we propose Mimer Trust, a solution to enhance the current TEE with secure and efficient access control and data sharing. We intend to allow data stored via the TEE to be accessed the same way as relational data in the normal world without compromising security. As illustrated in Fig. 2, our solution uses a Mimer SQL database in the normal world to provide rich functionality, including fine-grain access control - but to store the sensitive data within the TEE. This offers strong data-at-rest security and enables the TEE to enforce broad access policies, such as limiting write access to certain data tables and providing assurance against any normal world attack. The solution is lightweight and thus especially suitable for resource-constrained embedded systems. Our solution contains the following core components:

- A transactional storage library for a trusted application to persistently store data in the Trustonic TEE storage layer.
- A set of easy-to-use APIs for the trusted application in the secure world to manipulate secure data. With these APIs, data can be saved and fetched as relational records, in which a rich set of SQL standard data types can be used directly.
- An extension in SQL with a special view for the secured data in the secure world and a set of stored procedures to access these data. Client applications can select from these views and run these procedures in ordinary SQL statements through the Mimer SQL server running in the normal world. This also allows the joining of data from the database with data stored in the secure world, which helps achieve richer functionalities..
- SQL privileges control user access to the special views and execution of the procedures.
- Encryption of data in the normal world database server and client-server communication encryption. The encryption keys are also stored in the TEE.
- Essentially, it provides a two-layered security mechanism with strong protection for sensitive data and flexible access policies suitable for various application scenarios, with minimal impact on the overall performance.

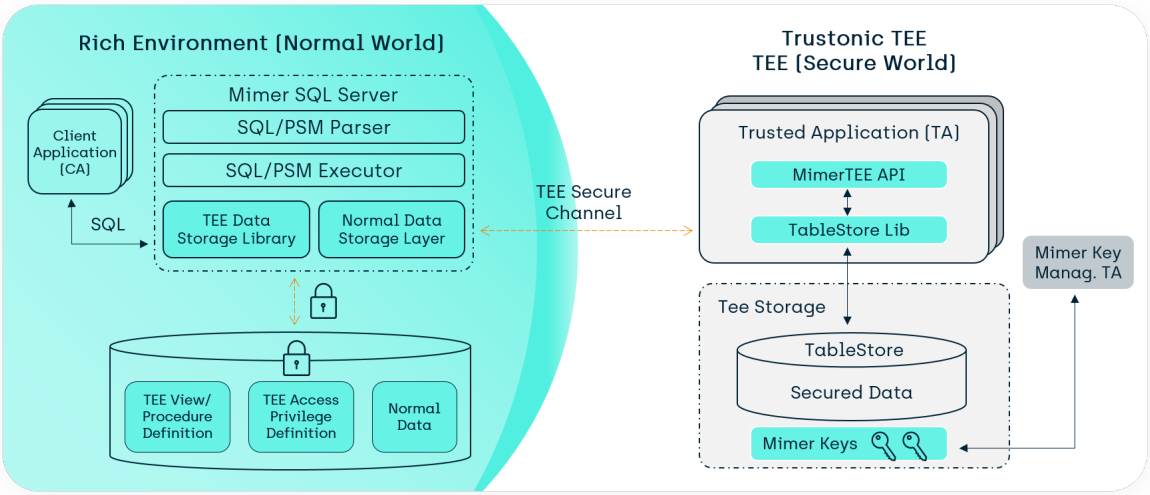


Fig.3 Architecture of Mimer Trust

2. The architecture of Mimer Trust

The overall architecture of Mimer Trust is shown in Fig. 3. In the secure world, trusted applications are equipped with the C MimerTEE API library, which provides interfaces to manipulate binary data as high-level relational records, and transactional storage capability.

The data stored in the secure world are mapped to SQL views and stored procedures generated automatically and efficiently by our tools. Trusted applications can define access privileges on these views and procedures for individual users. Client applications in the normal world with proper privileges can use these views and methods to fetch or modify the TEE data alongside normal SQL data.

Data communication between secure and normal worlds are made through secured channels governed by TEE, while data storage and communication in the normal world are protected with encryption.

MimerTEE API

As explained, all data in the TEE are stored in the secured storage area as binary files. The Secure Tablestore library is implemented to enforce atomic data storage to achieve reliable data retrieving and manipulation. Multiple data operations can also be grouped as transactions to prevent data inconsistency. Secure Tablestore maintains the simple key-value store, with an identification key and a binary string for each data object.

On top of the Secure Tablestore library is the MimerTEE API, a set of C routines. This set provides a high-level abstraction of the data as relational tables. They allow developers to create a table, define its structures, and manipulate the data for each column with explicit SQL data types. Code 1 shows the definition of a table LOGTAB in the schema SENSOR using MimerTEE API. It first declares a table descriptor, adds the columns, and creates the table. It then inserts a row into the table. The table is equivalent to the SQL table in Code 2.

Code 1. Create table SENSOR.LOGTAB and insert a row using MimerTEE API

```
// 1. Decleare table
MimerTeeTableRef table;
MimerTeeDeclareTable( "SENSOR.LOGTAB", &table);
// 2. Add columns
MimerTeeAddColumn("id",TEE_DTYPE_INT64,0,table);
MimerTeeAddColumn("sensor_id", TEE_DTYPE_INT64, 0, table);
MimerTeeAddColumn("msg",TEE_DTYPE_VARCHAR, 50,table);
MimerTeeAddColumn("log_timestamp", TEE_DTYPE_TIMESTAMP, 0, table);
// 3. Create the table
MimerTeeCreateTable(table);
...
// 4. Open table
MimerTeeOpenTable( "SENSOR.LOGTAB", &table)
// 5. Prepare a record
MimerRecord record;
MimerTeePrepareRecord(table, &record);
// 6. Set values to each column of the record
MimerTeeSetInt64(table, "id", 1, record);
MimerTeeSetInt64(table, "sensor_id", 1, record);
MimerTeeSetStringC(table,"msg","Warning",record);
MimerTeeSetTimeStamp(table, "log_timestamp", "2021-11-01 20:00:00.000000", record);
// 7. Do insert
uint64_t key = 0;
MimerTeeInsert( table, record, &key);
// 8. Free resource
MimerTeeFreeRecord(record);
```

Code 2. Equivalent SQL definition of SENSOR.LOGTAB

```
CREATE TABLE SENSOR.LOGTAB (
  id BIGINT,
  sensor_id BIGINT,
  msg VARCHAR(50),
  log_timestamp TIMESTAMP(6));
```

MimerTeeInsert, MimerTeeUpdate, MimerTeeDelete, together with MimerTeeSetXXX, set the values of columns and convert the relational record into the key-value pair. Similarly, according to the table definitions, MimerTeeFetch and MimerTeeGetXXX convert the binary TEE data into relational records. As can be seen here, a rich set of data types are available for the trusted applications within the TEE. Among the SQL standard data types, we currently support Boolean values, 32- and 64-bit integers, 32- and 64-bit floats, characters and national characters, binary data, and timestamps. Support for other data types is also planned. A concise dictionary is also created in the Secure Tablestore for the information of TEE tables and maintained automatically by the MimerTEE API library.

SQL for TEE Data

Our solution intends to allow client applications to access TEE data like normal SQL data. To achieve this, we introduce a unique view to represent the TEE data based on TEE Result Set Procedures and its underlying functions.

A TEE result set procedure extends SQL/PSM stored procedures that return a selected data set from the TEE data storage. Internally, the procedure uses functions to retrieve and modify TEE data. The procedure opens a session to the TEE storage and fetches all records from the table. Binary data are fetched from the TEE storage and converted to Mimer's relational format, which can be manipulated as regular records.

In Code 4, a view SENSOR.RESULT_SET_VIEW_LOGTAB is defined on such a result set procedure SENSOR.GET_RESULT_SET_LOGTAB(), which fetches SENSOR.LOGTAB data stored in the TEE and is created internally and automatically. Since this view is similar to a standard Mimer SQL view, we can create instead-of triggers to insert, update and delete the data in the base table (now the TEE storage) behind the view.

Code 4. A view for SENSOR.LOGTAB in the normal world based on a TEE result set procedure

```
CREATE VIEW SENSOR.RESULT_SET_VIEW_LOGTAB AS
SELECT * FROM TABLE(SENSOR.GET_RESULT_SET_LOGTAB());
```

A database user can then execute an insert towards the TEE data storage in the same way as other data, for instance, to join the view with another regular table:

```
SELECT * FROM SENSOR.RESULT_SET_VIEW_LOGTAB JOIN SENSOR_CONFIG ON SENSOR_ID,
```

or to add a new row into the TEE table:

```
INSERT INTO SENSOR.RESULT_SET_VIEW_LOGTAB VALUES(1, 1, 'Warning', LOCALTIMESTAMP).
```

When the TEE result set procedure is executed, data are retrieved and modified through the TEE Data Storage Library. This library converts data between binary and relational forms and communicates with the trusted applications through the TEE secured channel. The trusted applications then retrieve or manipulate the TEE data through the Mimer TEE APIs and return the results or acknowledgements to the server and clients.

A significant advantage of Mimer Trust is the controlled access to data from multiple client applications. Instead of client applications individually talking to a trusted application and requiring individual binding, the trusted application now only binds to the Mimer SQL server. The Mimer SQL server is responsible for authenticating authorising access from client applications to the data.

These views and procedures are created by the database administrator used by the firmware developer. The administrator can grant access privileges of these structures individually to individual database users. For instance, the administrator can grant the right to update SENSOR logs to the user LOG_ADMIN:

```
GRANT UPDATE ON SENSOR.RESULT_SET_VIEW_LOGTAB TO LOG_ADMIN
```

When a client application logs in as LOG_ADMIN and tries to update the view, the server checks the user's granted privileges and allows it to modify the data. Other unauthorised requests are denied. This is performed in the same way as ordinary privilege checks.

Data Encryption in Mimer SQL

Whilst the focus of Mimer Trust is to store data within the TEE, it is also useful to support regular tables in the REE that are protected via encryption. Mimer SQL uses AES encryption to encrypt the normal data stored on a disk. Data are encrypted when written to the disk and decrypted when queried and loaded into the system memory from the disk. The client-server communication is also protected using AES encryption. Since the strength of the encryption relies on the key to remain secret, it is essential to consider where the key should be stored.

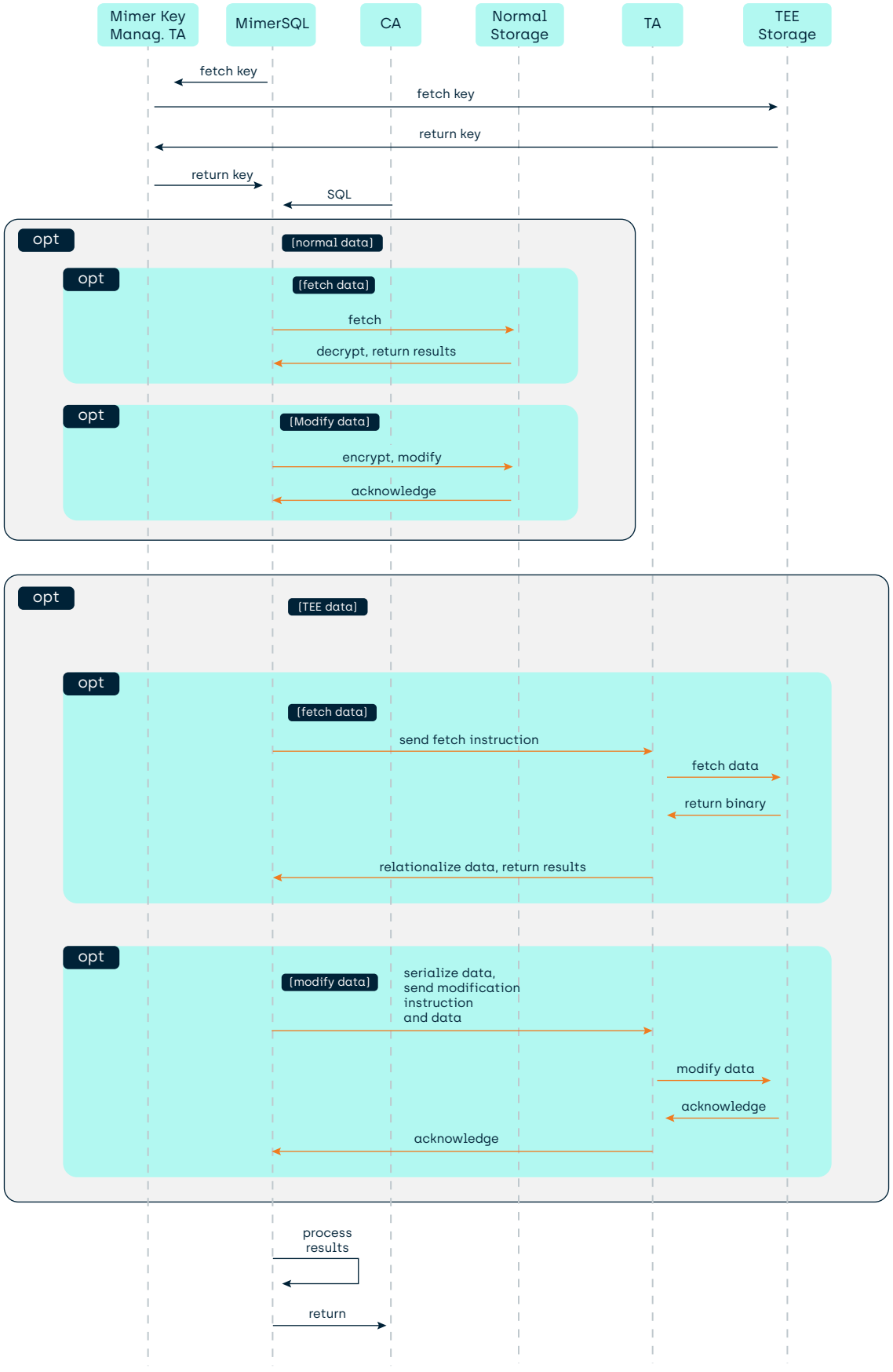
To solve the problem of database encryption key storage, we store our keys in the TEE storage, which is protected by hardware isolation. We implement a Key Management Trusted Application, a trusted application in charge of the keys for Mimer SQL. Keys are generated in the TEE by the TA. All code execution that uses the key, such as encryption and decryption, is executed by the Key Management Trusted Application running in the TEE. It is also completely isolated from other trusted applications in the TEE. This way, the key is protected from potentially malicious applications or code running from the normal world throughout its lifetime.

The Mimer SQL server opens a session with the Mimer Key Management Trusted Application when started and then keeps this session open, waiting for commands to encrypt or decrypt data.

Encryption is an essential link in the security scheme. Since both communication and the data dictionary, which contains the user and privilege information, are encrypted, and the keys are protected in TEE, malicious applications cannot steal user identities and bypass the access control. This ensures that our solution maintains the same level of security compared to the original TEE architecture.

The sequence of executing an SQL statement with TEE data is explained in Fig. 4. The statement can contain multiple normal tables and TEE results sets. The join results of the TEE and normal data are constructed in the Mimer SQL server. Since the database engine performing compilation and result formation is in the normal world, we maintain a minimal codebase. As a result, we require little resources in the TEE, a common requirement in embedded systems.

Fig. 4. Workflow of executing SQL in the normal world



Layered Security and Access Policies

The mechanics of Mimer Trust have been described in the sections above. By integrating the database privilege system with the TEE, Mimer Trust achieves two layers of security. The primary means of access control occurs within the Mimer SQL server, which supports a fine-grained access policy. In addition, the TEE-backed approach adds a stronger layer of access enforcement within the trusted execution environment, which is far harder for an attacker to breach.

The trusted application has absolute control over the secure data and provides several TEE-enforced access policies defined on a per-table basis. By modifying the trusted application's interfaces exposed to the Mimer SQL server, we can easily achieve the following policies suitable for various application scenarios:

Default

The Mimer SQL server is given full read /write access to the table and is solely responsible for enforcing user-based access policies from client applications.

Read Only

The Mimer SQL server has read-only access to data, which must be created from within the TEE.

Write Once

The Mimer SQL server can append new data and query existing data but cannot change data once added.

Wrap Around

A variation of write-once where records have a time-based key, and old records are purged automatically.

This additional layer of security makes life far harder for an attacker and far easier for a system to gain certification. This is because the integrity of the data storage remains entirely within the scope of the Trusted Execution Environment, which itself can be certified to a high level according to the Common Criteria standards [2] (for example, Trustonic's Kinibi is certified to EAL 5+ [3]). Standard compliance certification helps those new to a TEE to obtain additional confidence. Read Only data is useful where a trusted application generates the data - perhaps from a secure peripheral that the TEE can only access. Write Once data is effective for general logging of activity (for example, recording service history). In contrast, Wrap Around logging is useful for "black box" recording and can be sealed in the case of a "significant event".

Whilst the primary APIs to access data in Mimer Trust are SQL based, additional functions can be provided for vertical applications - for example, to provide encrypted/signed export of data directly from the TEE or to reset Read Only or Write Once tables. As Mimer Trust provides a rich API to trusted application writers, customisation has significant scope.

3. Integration Details for Embedded Application Development

Development Concerns

When developing embedded applications, it can be hard to see the states and the data the applications use. This can be even more of an issue when a TEE is involved because of the encapsulation for security. In traditional embedded programming, the only way to see the data used is to create a dump and copy it to the developer's workstation. This is both cumbersome and inaccurate as the dump is just a snapshot rather than the live data that the application sees.

To illustrate this, let us consider two situations in the development of the data box in Fig. 1. In one case, if the developer of the Logging application realises that the logs written to the storage are incorrect, he needs to write a program to extract the data from the storage and analyse it offline. Due to the nature of the TEE, it is tough, if not impossible, to debug the trusted application running on the device. Suppose the Viewer client, in the other situation, suspects errors in the data and wishes to analyse other data in the TEE. In that case, a new interface needs to be implemented in the Logging first to expose this data.

With Mimer Trust, exposing the live data during development becomes convenient. Since the data is accessible through Mimer SQL as a standard view, it is possible to use any database tool using standard APIs like JDBC, ODBC, or ADO.NET to see the data on the fly. Fig. 5 gives an example of using a third-party database authorisation tool to view the LOGTAB data stored in the TEE during development. A practical recommendation is to grant access to the secure data to a development user during implementation and testing and revoke the access before the system goes into production.

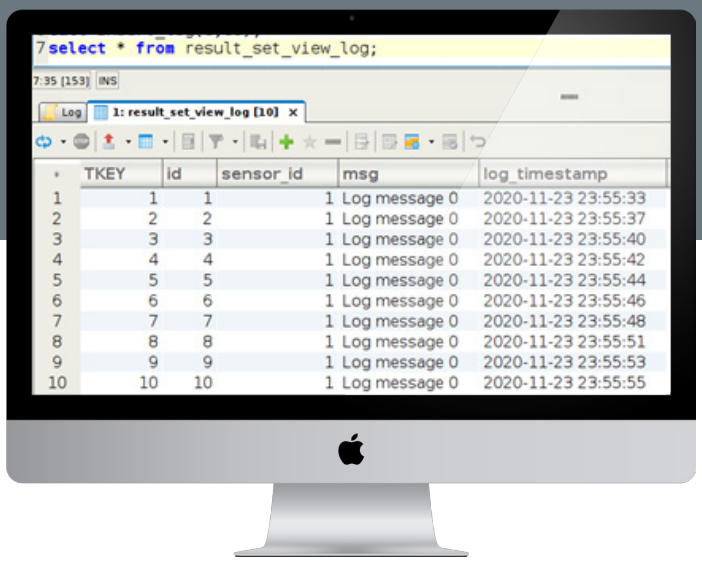


Fig. 5 Viewing TEE data during development

Deployment Concerns

A crucial question is how to deploy our solution in the embedded devices, especially integrating it into the existing secure booting sequence. In a TEE-enabled system, the secure world operating system is booted first, during which the firmware of the device is attested and verified. Then the normal world operating system is booted, followed by the start of the client applications. The booting sequence of a device with Mimer Trust contains a few extra steps, as shown in Fig. 6.

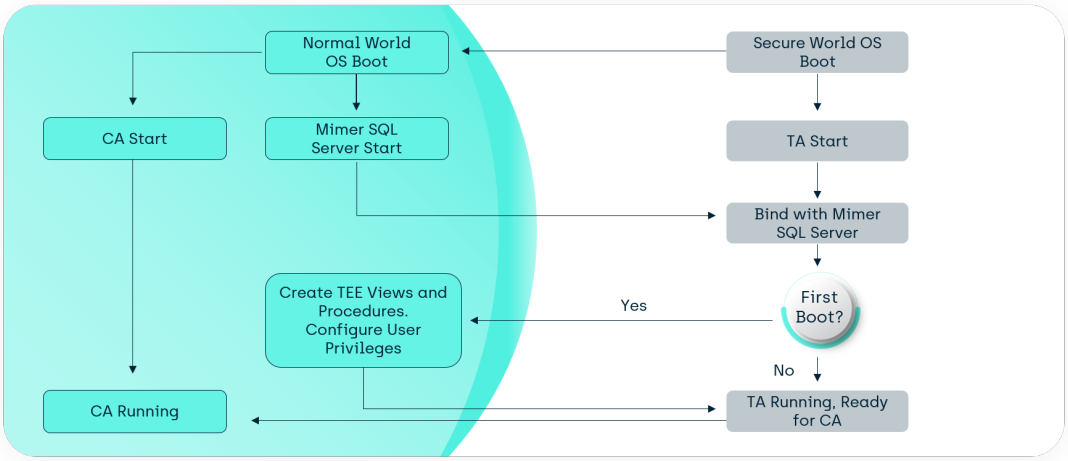


Fig. 6. Deployment and booting sequence with Mimer Trust

As in the ordinary booting process, the trusted application can create data that cannot be shared with the normal world. For the data to be shared, the trusted application only holds the initial configuration about the table structures and the user privileges.

After the secure world booting process starts the trusted application, the normal world booting process starts the Mimer SQL server; the server is bound with the trusted application through ordinary TA/CA binding. Suppose the current booting is the very first booting of the device. In that case, the booting process checks the trusted application's configuration, and the database administrator creates the TEE views, procedures, and user privileges. This step also creates the corresponding tables in the TEE storage, where the data resides.

After the client applications are started, they can connect to the Mimer SQL server with their given user identities and query or modify the data to which they have the corresponding privileges.

Our solution makes it easy to update the data access authorisation of a client application or add a new client application. Authorised client application developers will be provided with specific user identification to access and modify their intended data. Device manufacturers can also easily update their data sharing policies by updating the user privileges to the TEE views and procedures. As an update of trusted applications is often done with firmware updates, these views, procedures, and privileges can be reconfigured and reset, which is part of the ordinary firmware update process.

Note that the Kinibi TEE supports RPMB-based anti-rollback for secure storage so that rollback-based attacks against data stored in the TEE are not practical. In addition, TEE storage is device bound, meaning that it is encrypted with an SoC (System on Chip) specific key. This means that if an attacker gains the ability to clone encrypted storage, they cannot insert the data into a second device, even of the same model.

4. Performance Evaluation

We present a series of tests on an actual device to evaluate the performance of Mimer Trust. We use a HiKey 960 board equipped with a quad-core Arm Cortex-A73 CPU, a quad-core Cortex-A53 CPU, and 3 GB of memory. Arm Security Algorithm Accelerator is enabled, a common hardware block for accelerating encryption/decryption on ARM chips. However, to test the performance with restricted resources, the memory limit for each trusted application is set to be 16 KB stack and 64 KB heap, respectively.

Direct Data Access from Trusted Application

Our first test suite evaluates the performance of direct data access from the trusted application. We use the table SENSOR.LOGTAB is defined in Code 2 as the structure of the tested secure data object stored in TEE and perform the following tests:

Insert 1000 rows.
Insert 1000 logs into SENSOR.LOGTAB.

Select 1000 rows.
Fetch all logs from SENSOR.LOGTAB.

Update 1000 rows.
Fetch all logs from SENSOR.LOGTAB, and update each one.

Delete 1000 rows.
Delete all logs from SENSOR.LOGTAB.

To serve as a baseline, we have developed a trusted application in C that performs a series of operations on the secure data through vanilla Trustonic API without Mimer Trust. As a counterpart, we created another trusted application in the environment with Mimer Trust deployed, where all data access and manipulation are made through MimerTEE API. The performance (Fig. 8) shows that MimerTEE API introduces little overhead than the in-house solution and would not impact system performance requirements. Note that the TEE storage is encrypted so that the baseline performance will be less than unencrypted REE storage.

Fig. 8. Performance comparison for direct access from a trusted application

Data Access from Client Applications

Our second test suite examines the performance when data is shared between the trusted and client applications. In addition to the previous tests, we introduce two more tests, **Select Range** and **Join,** that examine the performance of ranged search and joins respectively. Select Range fetches all logs whose sensors id's are between 1 and 500. Join combines the log data in the TEE with the sensor's information stored in an ordinary table SENSOR_INFO in Mimer SQL. The table is defined as CREATE TABLE SENSOR_INFO (SID INT PRIMARY KEY, NAME CHAR(50)).

The baseline trusted application stores and fetches binary data objects in the TEE, while the client application encodes and decodes the data between binary objects and the readable form, using the in-house data structure. For the baseline counterpart, we implement a client application for the device with Mimer Trust installed. The client fetches and manipulates data from the TEE using the Mimer C API, which communicates with the Mimer SQL server using SQL statements. The SQL statements used by the client application are as follows:

- **Insert 1000 rows.** INSERT INTO SENSOR.LOGTAB (ID, SENSOR_ID, MSG, TIMESTAMP) VALUES (:id, :sid, 'LOG MESSAGE', TIMESTAMP '2021-11-01 01:01:01.123456').
- **Select 1000 rows.** SELECT * FROM SENSOR.LOGTAB.
- **Update 1000 rows.** UPDATE SENSOR.LOGTAB SET MSG = 'WARNING' WHERE SENSOR_ID > 0.
- **Delete 1000 rows.** DELETE FROM SENSOR.LOGTAB.
- **Select range.** SELECT * FROM SENSOR.LOGTAB WHERE SENSOR_ID <= 500.
- **Join.** SELECT * FROM SENSOR.LOGTAB JOIN SENSOR.SENSOR_INFO ON (SENSOR_ID = SID).

Fig. 9. Performance comparison for data access from the client application

The performance results are shown in Fig. 9. The increased time in Insert, Update and Delete is expected, as several steps are added to share data, for instance, the compilation of the statements and the conversion of the data. Select and Select Range, on the one hand, have seen double response time compared to the baseline. On the other hand, both operations are very fast and, therefore, more sensitive to the overhead introduced by the extra steps.

One thing to note is that an improved response time has been observed in Mimer Trust for the Join of secure and normal data (Fig. 9(f)). This is particularly interesting given the fact that Mimer Trust introduces a longer time for simple selects of only TEE data. This suggests that Mimer Trust can bring substantial benefits for scenarios of advanced data sharing between the secured and normal worlds, which is exactly the main problem we target.

Encryption Overhead

To examine the performance impact of performing encryption and decryption through Mimer's Key Management trusted application, several tests were performed on the HiKey-board. The tests are described in Table 1. The tests included inserting 5000 rows into a table from Mimer SQL in the normal world, using different methods.

Tab. 1. Tests to evaluate encryption overhead

Test	Description
Test 1	Inserting 5000 rows without prepared statements
Test 2	Inserting 5000 rows using prepared statements with implicit transaction handling
Test 3	Inserting 5000 rows using batched statements and auto-commit
Test 4	Inserting 5000 rows using batched statements and explicit transaction handling. Transaction size: 100

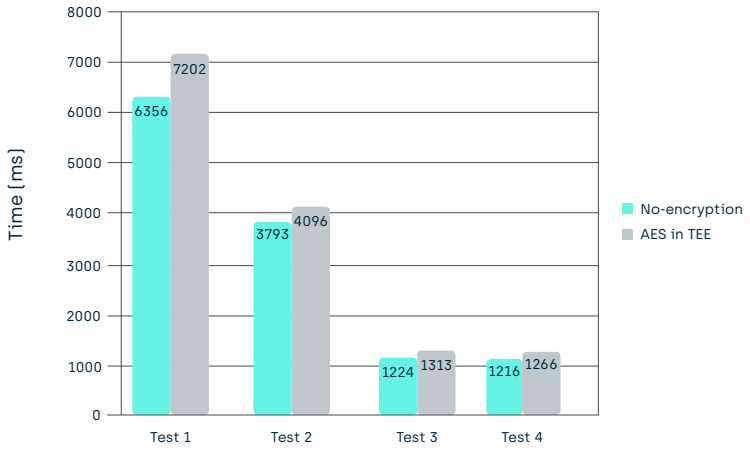


Fig. 10 Performance comparison for encryption overhead

The results of the tests can be seen in Fig. 10. The results show an overall minimal impact on the execution time when utilising the encryption and decryption when the keys are kept secure in the TEE. Only in Test 1, the performance difference is more noticeable. In Tests 2, 3, and 4, prepared and batched statements were used, resulting in minimal time differences.

5. Conclusion

In this paper, we have presented our solution Mimer Trust for efficient and secure data sharing between the secure and the normal worlds in a TrustZone based embedded system. Our solution is based on the synergy of Trustonic TEE and Mimer SQL, which integrates modern database technology into the data management of the state-of-practice TEE framework. Mimer Trust allows client applications to access secure binary data through the rich SQL interface, using our TEE result set procedures and view. Secure data access can be easily configured and controlled through the proven database user privilege control framework in Mimer SQL. We also provide low-level access control within the TEE itself to allow high assurance for data integrity – for example, for evidential use.

The communication between trusted applications and Mimer SQL is protected by TrustZone isolation. The communication between client applications and Mimer SQL and the database dictionary containing the TEE information is protected by encryption. As a result, direct access to data from unauthorised clients or malefactors is impossible. Instead, REE access is always via the SQL server, where flexible, fine-grained data access control can be achieved. Both trusted and client applications can manipulate secure data via high-level programming interfaces with rich data types. This also reduces implementation and debugging efforts during embedded software development. Our solution requires minimal resources in the TEE, which is ideal for resource-constrained embedded systems.

Our solution improves joining data in the TEE with ordinary relational data, which makes data sharing between the two worlds more efficient. This opens up a wide range of new OEM applications and services while providing strong compliance for current and future privacy and security regulations. Although some overhead is introduced for other evaluated data operations, its influence is small and is outweighed by its benefits, inefficient data sharing, reduced development costs, and enriched application scenarios.

About Trustonic Kinibi TEE

Trustonic Kinibi is a GlobalPlatform certified TEE based on ARM's TrustZone technology and is installed in more than 2 billion embedded devices. TrustZone enables two environments to run alongside each other with a high degree of isolation. The Regular Execution Environment (REE) is where the normal embedded operating system and applications reside, such as Linux, Android or QNX. The TEE comprises a trusted operating system and trusted applications (TA). Trustonic Kinibi TEE is certified to Common Criteria Standards. It runs all security-sensitive applications such as key storage and management, biometric processing, or Digital Rights Management (DRM) for video playback.

About Mimer SQL

Mimer SQL is a cross-OS and platform-independent relational database system used in mission-critical systems, from embedded systems to enterprise servers. These deployments cover a variety of platforms, including small-footprint installations for ARM-based embedded systems and devices. Mimer SQL complies with the SQL and PSM (Persistent Stored Module) standards and offers several Application Programming Interfaces (API) for clients, including JDBC, ODBC, ADO.NET, Mimer C/C++ API, MimerPY for Python, among others. In addition, table-level data access privilege support is provided for controlling user access of data and execution privileges for stored procedures, functions, and modules.

200 Cambridge Science Park
Cambridge CB4 0GZ
United Kingdom

info@trustonic.com

Kungsgatan 64
SE-753 41 Uppsala
Sweden

info@mimer.com