CertainID

Seyit Camtepe, Seung Jang, Garrison Gao

August 2019

Kollakorn Corporation Ltd

David Matthews, Chief Executive Officer

Commercial-in-confidence

Citation

S. Camtepe, S. Jang, G. Gao (2019) CertainID Project. CSIRO, Australia.

Foreword

CertainID is a CSIRO Kick-Start project with Kollakorn. CSIRO Kick-Start is a new initiative for innovative Australian start-ups and small SMEs, providing funding support and access to CSIRO’s research expertise and capabilities to help grow and develop their business.

Kollakorn and CSIRO Data61 collaborate to develop an innovative method for secure use of biometric features for signing electronic transactions. CertainID does not store any biometric information associated with individuals, and it ensures that individuals’ mass biometric data is not lost or compromised if the device storing CertainID data is lost or compromised, e.g., no sensitive information associated with individuals can be hacked off the device.

Contents

Figures

Tables

Acknowledgments

Following people have contributed to the CertainID project so far:

- **Data61**: Seung Jang, Garrison Gao, Seyit Camtepe, Surya Nepal
- **Kollakorn**: David Matthews
- **Infitecs**: Gary Fitzgerald
- **Infinity Optics**: Raghavendra Prasad NL, Gilles Mathieu, Alfred Chan, Michael Gui

Executive summary

Biometrics (i.e., something you are) constitute an essential factor in multifactor authentications (i.e., something you are, something you are given and something you know). The existing biometric solutions store and communicate biometric templates, a digital representation of unique biological features such as fingerprints, palm, palm veins, hand geometry, face, iris, retina, DNA and odour, and behavioural characteristics such as typing, gaits and voice. The fundamental problem with these biometric templates is that their compromise causes permanent harm. The stolen biometric template can be used to claim the identity of the victim continually - a victim may be able to change some biometric features through a series of major plastic surgery or organ transplant, but DNA-based biological features cannot be changed as they are stored in each of 100 trillion cells of the human body. These challenges make biometrics-based security solutions (e.g., authentication, confidentiality and digital signature) risky for both users from whom these features are extracted and organizations which extract, use and store these features.

The CertainID project fills a significant gap by providing a safe method for biometrics-based security solutions. The project has started with the investigation of a white paper and two Kollakorn patents: A Biometric Authentication System (WO 2013/134832 A1) and Biometric Identification Method (WO 2010/132928 A1). The core idea of patents which is transferred to the CertainID solution is to use biometric information to lock and unlock revokable security keys and use these keys to secure transactions. Hence, CertainID eliminates the need for storage of any biometric information.

CertainID uses biometric features through a one-way mapping to create a revokable master key and uses the master key to encrypt a set of randomly generated private keys. Users present their biometric features to unlock a private key for signing (or decrypting) a transaction, e.g., bank transaction, an electronic form, email, course registration, project delivery, exam and attendance. The user has the freedom to revoke and renew these public/private keys anytime they like. Even when an adversary obtains these encrypted key pairs, they can obtain no information on users' biometric features.

CertainID helps remove the security key management burden and associated risks both from individuals and organizations. CertainID ensures that individuals' mass biometric data is not lost/compromised, and no sensitive information associated with individuals can be hacked off the device if the device storing CertainID data is lost or compromised.

1 Introduction

Biometrics (i.e., something you are) constitute an essential factor in authentication. There are three factors:

1. something you are (biometrics),
2. something you are given (hardware or software tokens), and
3. something you know (secrets and passwords).

Multi-factor authentication schemes are built by the following combinations of these three factors:

a) One-factor authentication, i.e., any one of the factors
b) Two-factor authentications, i.e., any two of the factors
c) Three-factor authentications, i.e., all three factors

Among these three factors, “something you are” (biometrics) are the most challenging but at the same time the most usable one. Usability comes from the fact that a user presents themselves – they do not need to remember a password or carry hardware around. The challenging aspect has many contributors such as the difficulty of developing the technology (i.e., biometric sensor) to extract biometric features in the most usable way and ensuring that these features won’t end-up at malicious hands - this project focuses on the security of the biometrics.

The existing biometric solutions store and communicate biometric templates, a digital representation of unique biological features such as fingerprints, palm, palm veins, hand geometry, face, iris, retina, DNA and odour, and behavioural characteristics such as typing, gaits and voice. The fundamental problem with these biometric templates is that their compromise causes permanent harm. The stolen biometric template can be used by malicious people to claim the identity of the victim continually - a victim may be able to change some biometric features through a series of major plastic surgery or organ transplant, but DNA-based biological features cannot be changed as they are stored in each of 100 trillion cells of the human body. These challenges make biometrics-based security solutions (e.g., authentication, confidentiality and digital signature) risky for both users from whom these features are extracted and for organizations which extract and store these features, i.e., an organization losing control of biometric information of its customers can face severe legal and reputational consequences.

The CertainID project aims to make it safer for users to supply their biometric information and for organizations to build user-friendly solutions composed of strong biometrics-based security. CertainID achieves this aim is by changing the way biometric features are used – information that can be used to extract the exact biometric features should not be stored.

The CertainID project has started with the investigation of one Kollakorn white paper and two Kollakorn patents: A Biometric Authentication System (WO 2013/134832 A1) and Biometric Identification Method (WO 2010/132928 A1). The core idea of patents which is transferred to the CertainID solution is to use biometric information to lock and unlock revokable security keys, store only encrypted security keys and use these keys to secure (i.e., sign or encrypt) application transactions, as illustrated in Figure 1. Hence, CertainID eliminates the need for storage of any biometric information.

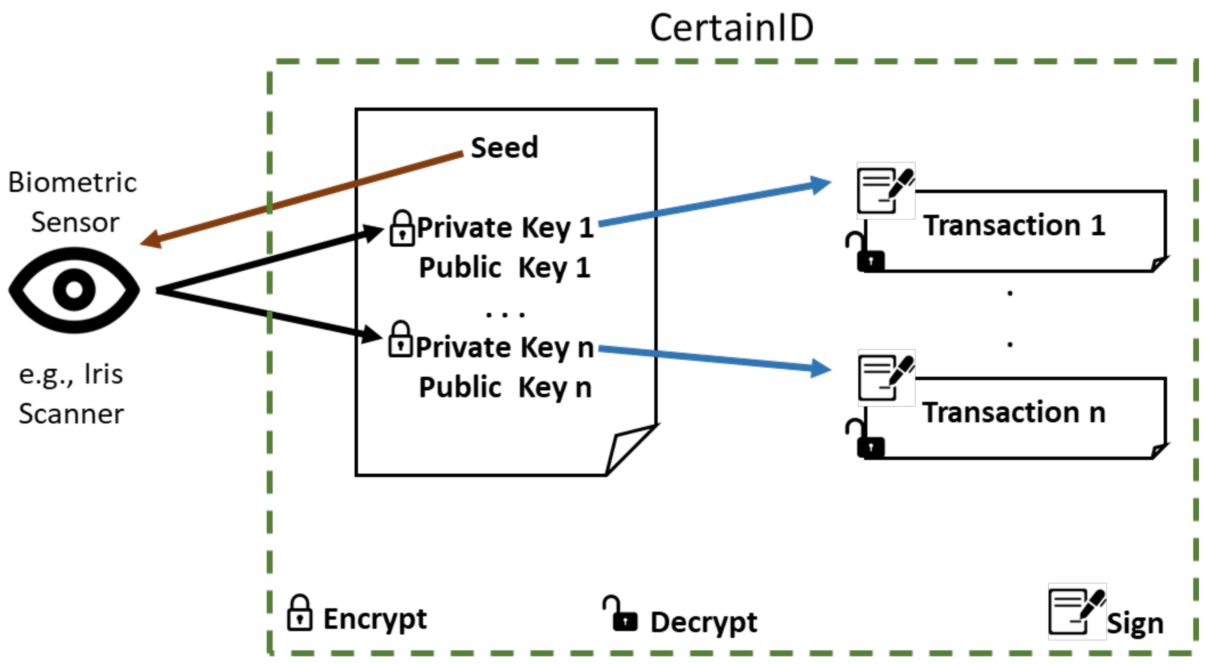


Figure 1 CertainID Concept. A sensor challenged with a seed reads a biometric feature (e.g., iris scan) and produces a biometric code. Different seeds result in different biometric codes. Biometric code is used to encrypt private keys. Every time a private key is required for signing/decrypting a transaction, the sensor should be invoked with the same seed to extract the matching biometric code and release the private key temporarily.

CertainID uses biometric features through a one-way mapping to create a revokable master key and uses the master key to encrypt a set of randomly generated private keys. Users present their biometric features to unlock a private key for use in signing (or decrypting) a transaction, e.g., course registration, project delivery, exam and lab attendance in education domain. Verification of signed transaction can be done by using the public key available to anybody. Similarly, a transaction that should be viewable only by the user can be encrypted by anybody using the public key of the user. User has the freedom to revoke and renew these public/private keys anytime they like. Even when an adversary obtains these encrypted keys, they can obtain no information on users' biometric features and the victim can easily reset the compromised keys.

CertainID helps remove the security key management burden and associated risks both from individuals and organizations. Users can safely present their biometric information knowing that they will not be stored on any devices. Organizations only needs to store encrypted security keys which are not useable without the Biometric data and which can easily be revoked in the worst-case scenario – this makes organizations' job easier to comply with security and privacy regulations. CertainID achieves so while ensuring that individuals' mass biometric data is not lost/compromised, e.g., if the device storing CertainID data is lost or compromised, no sensitive information associated with individuals can be hacked off the device.

2 CertainID Key Management and Signature Scheme

2.1 Keys and Cryptographic Primitives

Secure and usable methods to identify Internet users have been a critical problem for Internet applications. Password-based methods (i.e., something you know) have proven to be vulnerable as people tend to pick easy to remember passwords, use the same password for many Internet applications or share them with others. The similar problems exist for Internet applications and protocols that depended on strong cryptographic methods. No matter how strong these cryptographic methods are, they become vulnerable when their secrets are not protected well.

CertainID is a biometrics-based key management solution. However, combined with a biometric sensor, it can be considered as a suite that solves identification, authentication and key management problems.

- CertainID uses revokable biometric codes to protect (lock/unlock) arbitrary number and type (symmetric or asymmetric of different sizes) of secrets as illustrated in Figure 1. The keys are mainly used to ensure integrity, authenticity and confidentiality of transactions supplied by client applications.
- CertainID can be used to authenticate a user if they can supply the matching biometric code to unlock a set of secret belonging to the claimed ID.
- CertainID can be used to identify a user if the biometric code unlocks a set of secrets belonging to a known ID.

CertainID expects to receive a minimum of 128-bit biometric code from the biometric sensor. It is critical to have high-entropy biometric codes. This can be ensured by the sensor by using a privacy amplification method such as hashing or encrypting using a salt value. Biometric code is used to encrypt user's secrets (e.g., private keys in Figure 1) with 128-bit Advanced Encryption Standard (AES) algorithm.

Selection of key size is an important aspect of security. Any cryptographic scheme having less than an 80-bit security rating is considered as weak. 80-bit security means that a computer needs to search 2^{80} different keys in a brute force key attack to break the code. State-of-the-art computing systems can achieve a brute force key attack for the algorithms having less than 80-bit security. AES-128 (AES with 128-bit key size) provides 128-bit security scale, which is assumed to be safe. Additionally, it has been shown that brute-force key search on quantum computers cannot be faster than $2^{n/2}$, which means that AES-256 (AES with 256-bit key size) provides 128-bit security scale against quantum brute force key attack. Table 1 by NIST compares the security scale of the symmetric and asymmetric key schemes.

Table 1 NIST Key Size Guideline. CertainID enforces security scale to be higher than 80-bit.

Security Scale	Symmetric Key Cryptography Key Sizes	RSA and Diffie-Hellman Key Sizes	Elliptic Curve Cryptography Key Sizes
80-bit	80-bit	1024-bit	160-bit
112-bit	112-bit	2048-bit	224-bit
128-bit	128-bit	3072-bit	256-bit
192-bit	192-bit	7680-bit	384-bit
256-bit	256-bit	15360-bit	521-bit

The secrets (e.g., private keys in Figure 1) managed and protected by CertainID is generated using RSA-2048, which means 2,048 bits - 617 decimal digits. RSA-2048 may not be factorizable for many years to come unless considerable advances are made in integer factorization, quantum computing or computational power soon. It is possible to use 224-bit or larger elliptic curve cryptosystem as most signature schemes support curves at these sizes.

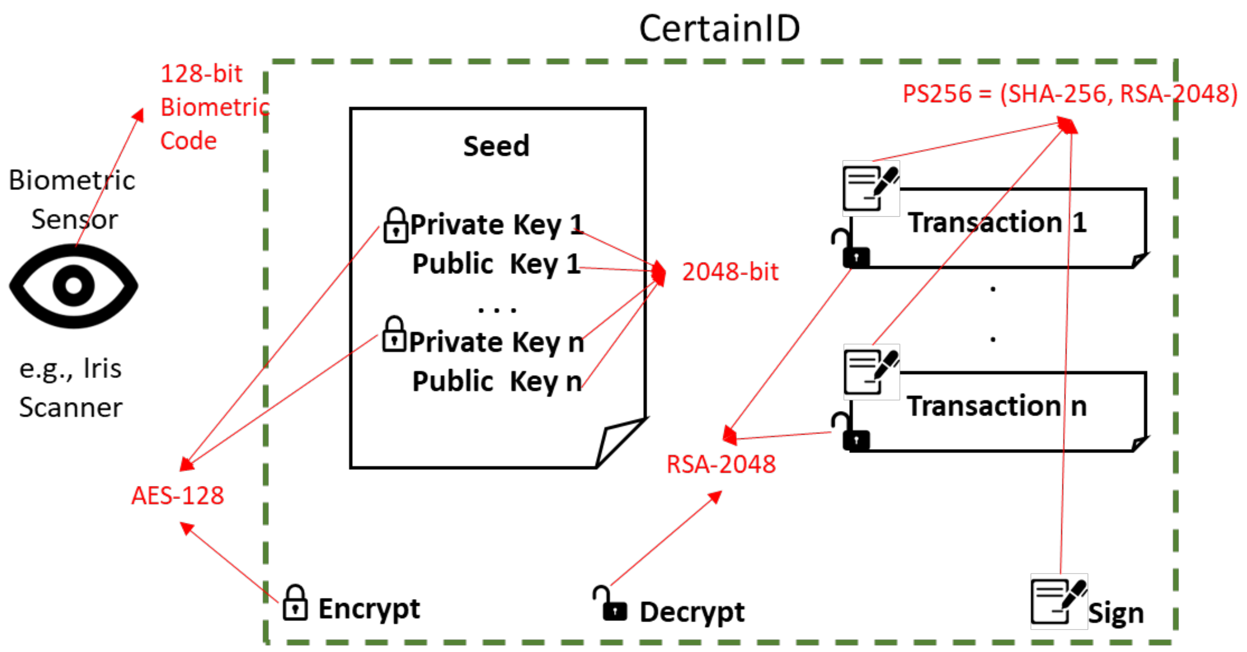


Figure 2 CertainID Keys and Primitives.

CertainID uses SHA-256 to reduce an arbitrary size transaction into a 256-bit hash. Hash of the message is encrypted by using 2048-bit private key and RSA-2048, following the PS256 (RSAASSA-PSS) digital signature scheme. The encrypted hash forms the signature for the transaction. It is infeasible to change this transaction as any change (e.g., a bit flip) will yield a different hash which will not match the encrypted hash in the signature. Such a change requires an adversary to break the RSA-2048 encryption.

CertainID can be used for confidentiality of the transaction as well. Any party knowing the public key of the user can encrypt a transaction using AES-128 and a random 128-bit symmetric key, encrypt the symmetric key by using RSA-2048 and 2048-bit public key. Only the user providing the correct biometric code can unlock the private key which can be used to decrypt the symmetric key and decrypt the transaction. Due to the computation complexity of the 2048-bit encryption, it is not practical to use RSA-2048 like algorithms to encrypt arbitrary sized transactions directly.

2.2 Biometric Sensor

CertainID is designed to work with any desired biometric sensor (e.g., fingerprints, palm, palm veins, hand geometry, face, iris, retina, DNA and odour, and behavioural ones such as typing, gaits and voice) which can accept seed and produce a 128-bit biometric code with high entropy. The sensor should provide the same biometric code every time the same seed is presented. The sensor may randomly generate the seed during the initial registration phase and supply it along with the biometric code. CertainID stores this seed along with the secrets of the user. Later, every time CertainID presents the seed, the sensor should supply the same biometric code. The sensor should be generating a different (random) seed each time a user is revoked or newly registered.

This project uses iris sensor/camera supplied by a Kollakorn partner, Infinity Optics. This is a lightweight camera which can be integrated into laptop or tablet scale computers or provided as a USB/Bluetooth peripheral device for desktop computers and smartphones. The camera can give stable iris reading from a reasonable distance, resolving many usability issues surrounding the iris biometrics-based systems. CertainID communicates with the camera through API calls supplied by Infinity Optics.

The internal workings of the camera and the algorithm that Infinity Optics uses to extract the biometric code are intellectual properties of Infinity Optics and kept confidential. For this reason, we explain our proposed iris code extraction process and CertainID's requirements on biometric codes in this section.

2.2.1 Biometric Code Extraction

The aim of iris code extraction is to convert the iris into binary code, which will be used for cryptographic key binding (Figure 1). Figure 3 illustrates the steps involved for extracting the iris binary code. The first step is to detect iris boundary. Both inner (pupil) boundary and outer (sclera) boundary are located through Daugman's method.

The second step is normalisation, as shown in Figure 3 (b). This is because irises of different people may be captured in various sizes (same eye may also vary due to pupil dilation); therefore, normalisation is employed to normalize them to be the same size to achieve more accurate recognition. Afterwards, Daugman rubber sheet model is utilised to unwrap the iris region into a rectangular block.

The last step is using Gabor filter to encode the iris into binary `1'/`0'. Gabor filter (by Dennis Gabor), is a linear filter used for texture analysis – identifies specific frequency contents in specific directions in a localized region.

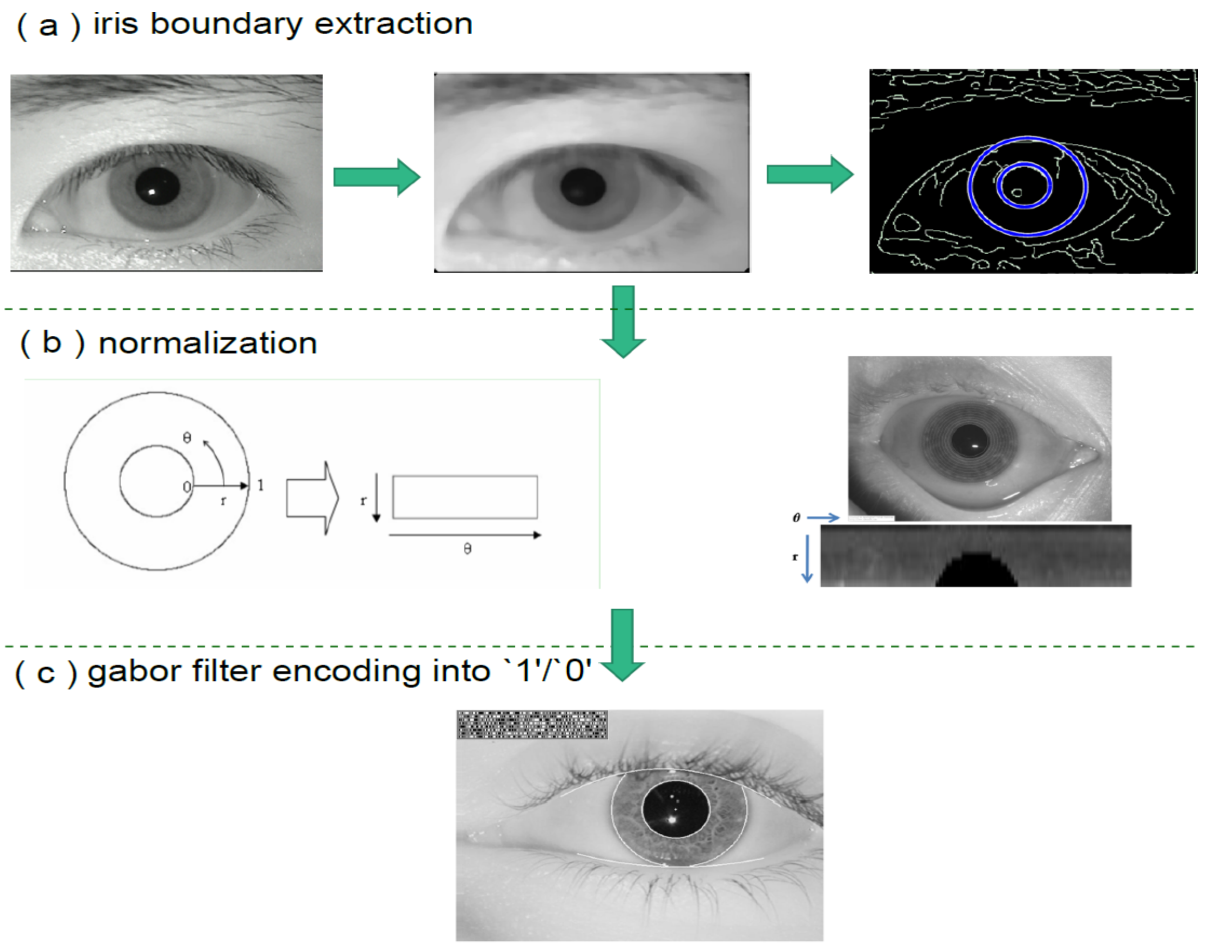

(a) iris boundary extraction
(b) normalization
θ
0
1
r
r
θ
θ
r
(c) gabor filter encoding into `1'/`0'

Figure 3 Iris binary code extraction.

2.2.2 Proposed Stable Key Generation Method

The iris binary code (binary string) is utilized for generating a stable biometric code, ***K***. We envision that this method (or equivalent) is embedded into a biometric sensor to generate a stable code. Noise in biometric authentication and recognition applications can be tolerated to some extent; however, not a single error can be tolerated in cryptographic key generation applications. We apply error correction code (ECC) to reconcile errors.

Figure 4 illustrates the enrolment and regeneration of a stable biometric code, ***K***. During the enrolment, a key K is randomly selected, potentially using a seed which is randomly generated during enrolment or supplied by CertainID during regeneration. Then ***K*** is encoded. We propose using concatenation code consists of two codes: ***BCH(n,k,t)*** code and repetition code ***REPET(m)***. Supposing we take ***BCH(255,45,43)*** code and ***REPET(9)***. Then we can split a 135-bit key into three slices, and encode each 45-bit key through a ***BCH(255,45,43)*** block. We get a codeword length of 255-bit. Because we have three 45-bit key, therefore, totally, after BCH encoding, we have 255*3-bit. Then we apply ***REPET(9)*** encoding to get 255*3*9-bit that is 6885-bit. This 6885-bit string is XORed with the $\boldsymbol{iris_{ref}}$ that is also 6885-bit to produce $\boldsymbol{iris_{lock}}$. Only the $\boldsymbol{iris_{lock}}$ is stored. Notably, the ***K*** is stored nowhere, it is discarded after the enrolment. The attacker can neither infer the $\boldsymbol{iris_{ref}}$ nor the ***K*** based on $\boldsymbol{iris_{lock}}$.

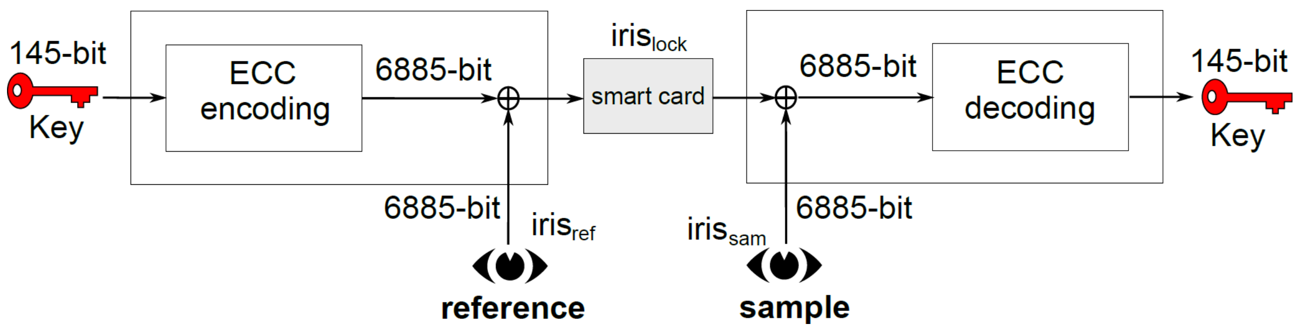


Figure 4 Biometric key binding based on iris. ECC is the abbreviation of error correction code.

During key regeneration, the user must be presented in front of the camera and produce iris$_{sam}$. If the distance between the iris$_{sam}$ and iris$_{ref}$ is smaller than a threshold, e.g., 20%, the ECC decoding can always correctly recover the enrolled ***K***. Otherwise, the ***K*** restoration will fail.

To help check whether the ***K*** has been correctly recovered, during the enrolment, a hash value of ***u_1 = HASH(K)*** is generated and also stored in the smart card. During the regeneration, after a trial of recovering the ***K***, a hash value ***u_2 = HASH(K)*** can be generated as well. If ***u_2*** = ***u_1***, then it means the ***K*** has been correctly recovered. There are certain properties worth to be mentioned:

- The ***K*** is randomly chosen, before binding, it has no link with any user.
- Due to the above reason, if the user's token is missing, a new token can be assigned to the user while also binding a different ***K***. So, the ***K*** can be flexibly reset.

The ***BCH(n,k,t)*** code is able to correct **t** errors within **n** bits. The ***REPET(m)*** is able to correct up to (m-1)/2 errors with m is always an odd integer. In general, the BCH code serves as inner correction and ***REPET(m)*** is an outer correction. Table 2 gives some ECC configurations to achieve certain false rejection rate given the intraFHD of the iris code. IntraFHD is intra Hamming distance between iris codes produced by the same eye from the same user. It is obvious that a lower intraFHD is always desired because it facilitates the error correction and requires shorter iris code.

BCH(n,k, t) code	number of BCH block	length of K	REPET(m)	length of iris	intraFHD (%)	FRR
(255,45,43)	3	135	7	5355	20	$< 10^{-9}$
(255,45,43)	3	135	7	5355	30	0.0564
(255,45,43)	3	135	9	6885	30	5.87×10^{-4}
(255,71,29)	2	142	5	2550	20	4.03×10^{-4}
(255,71,29)	2	142	5	2550	25	0.4472
(255,71,29)	2	142	3	1530	15	8.9×10^{-4}
(255,71,29)	2	142	3	1530	17	0.0271

Table 2 False rejection rate (FRR) as a function of varying error correction capabilities (ECC code configurations).

2.3 CertainID Cryptosystem

Once a stable biometric code is received from the biometric sensor Figure 5(a): (i) if the user is enrolling/registering, a public/private key pair is generated, and private key is encrypted using the biometric code, and (ii) if the user willing to sign a transaction or decrypt an encrypted transaction, the private key is decrypted temporarily. Any party can access the user's corresponding public key. It is worth to mention again that the biometric code is stored nowhere, but it is derived on-the-fly only when the user presents his/her iris.

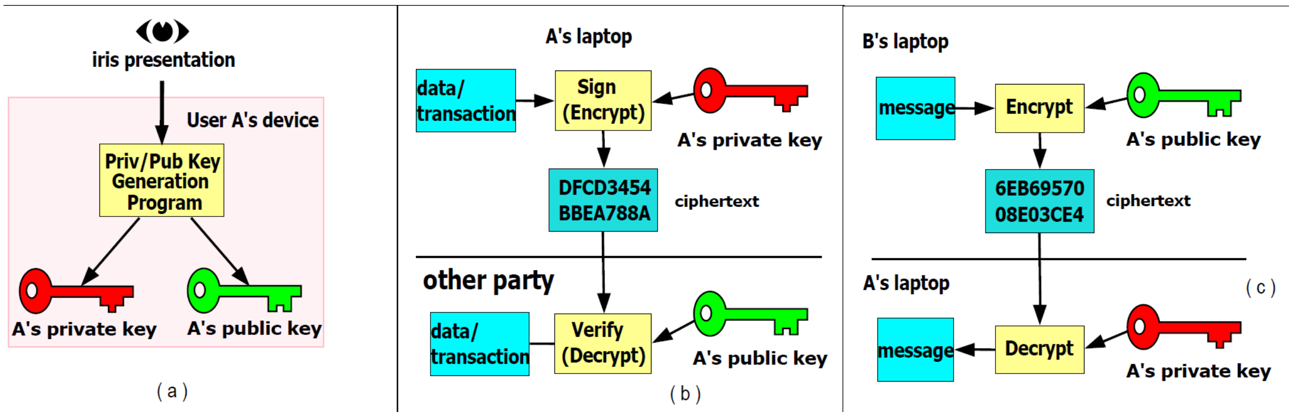


Figure 5 IRIS based public key infrastructure. (b) and (c) are based on white paper and patents of Kollakorn.

The signature scheme is illustrated in Figure 5(b). Most importantly, because, during set-up, the private key is bounded to the user, once the transaction is verified, it also means that this transaction is approved (performed) by the user. Only the user can unlock that private key to sign the transaction. So, the user cannot deny such a transaction.

The decryption scheme is illustrated in Figure 5(c). Herein, user B wants to transfer a message that is only allowed to be open by user A. User B firstly obtains user A's public key and encrypted the message, then sends the encrypted message (ciphertext) to user A. User A can open it only when he/she presents his/her iris to unlock the private key for decrypting the message.

There are alternative ways to use the biometric code. Rather than using the biometric code for encrypting the private key, it can be used directly as the private key. CertainID system does not store the private key and the user provides their biometric reading as needed. It should be noted that this method requires a biometric code with enough size for a public key cryptographic system such as RSA and Elliptic Curve as listed in Table 1.

CertainID cryptosystem has two phases: enrolment/registration phase and regeneration phase. The user must present their biometric information in each phase.

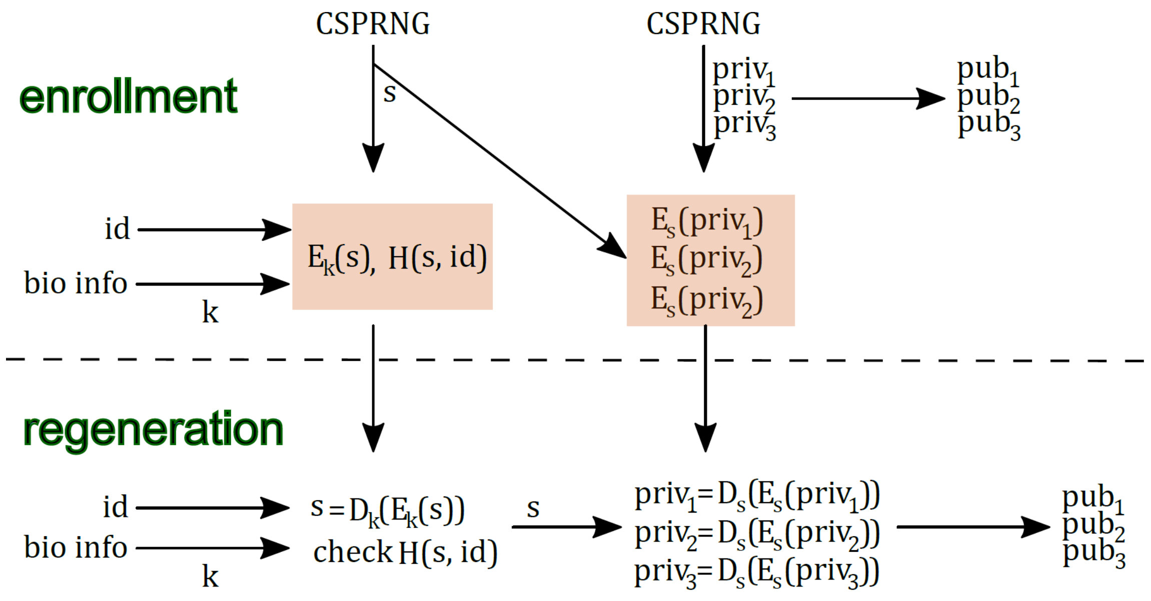


Figure 6 Biometric code based crypto-system flow. Only the shaded information is required to be stored. E()/D() stands for encryption/decryption function, H() stands for hash function.

Enrolment: Given a biometric code (biokey), ***K***, it is utilized to encrypt a secret ***s***. The encrypted ***s***, ***E_k(s)*** and the hash value of ***s***, ***H(s, id)*** will be stored after enrolment (Figure 6). The ***s*** is randomly generated from a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG). The reason of using ***s*** rather than directly employing ***K*** is that the secret ***s*** can be conveniently revoked.

Then the CSPRNG is used to generate several private keys. Here, three private keys, ***$priv_1$***, ***$priv_2$***, ***$priv_3$*** are taken for examplified demonstration. Different private key can be used for different purposes, e.g., the ***$priv_1$*** can be for email communication, while the ***$priv_2$*** can be for student exam checking-in. All these private keys are not stored in plaintext but encrypted by the secret ***s*** that can only be recovered when the biometric code is correctly presented. Based on public key generation algorithm, public keys ***pub_1***, ***pub_2***, ***pub_3*** are generated and made publicly accessible.

Regeneration: When the biometric information is presented after enrolment, the biometric code (biokey) ***K*** is extracted on-the-fly (Figure 6). It is utilized to decrypt ***E_k(s)***. To check whether the ***s*** is correctly decrypted, the decrypted ***s*** and input user ***id*** is hashed and compared with the enrolled hash value. If they are same, then the ***s*** is confirmed to be correctly recovered by the legitimate user ***id*** presence. The operation proceeds.

According to the specific application, e.g., email, a corresponding private key ***$priv_1$*** is reconstructed. If the application needs public key ***pub_1*** it will be produced as well given the chosen public key generation algorithm.

The public crypto library or module is planned to be integrated. We have checked the AES-128, SHA-256 and RSA-2048 are working in python by importing Crypto module.

https://github.com/garrisongys/pythonCrypto/blob/master/CryptoPythonExample.ipynb

2.4 CertainID Security Assessment

This section provides a high-level security evaluation for CertainID cryptosystem by focusing on the cryptographic primitives, communicated information (i.e., in transit between biometric sensor, CertainID and CertainID database), stored information (i.e., at rest at CertainID database), computed information (i.e., at the memory), and software vulnerabilities.

Cryptographic Primitives: CertainID uses the best practice cryptographic primitives with NIST recommended key sizes, e.g., AES with 128-bit keys, SHA256, RSA2048/ECC256, PS256. It is recommended that any updates on these primitives are monitored and depreciated ones are replaced with recommended ones by using the best practice crypto libraries.

Communicated Information: Sensitive information is communicated between Biometric Sensor, CertainID solution and CertainID database. It is assumed that the Biometric Sensor and CertainID solution are hosted on the same computer. Database can be on the same computer or on a network server for distributed applications. Biometric Code and Seed are communicated between Biometric Sensor and CertainID solution. A resident malware may be able to extract image of memory with perfect timing to leak this information. However, the leakage of this information is not a risk as the user can revoke this code anytime they like, and it is recommended that users regularly revoke and enrol a new set of keys using new biometric codes. As the additional layer of protection, appropriate firewall and intrusion protection combined with disk encryption should be considered as mandatory for the computer running the CertainID solution. Communication between CertainID solution and database should always be made over an encrypted channel such as TLS.

Stored Information: Encrypted sensitive information is stored in an encrypted database. Biometric code is not stored but used to encrypt a random key, ***s***. Hence, biometric data is safe. Private keys are encrypted by using ***s***. Once stored, this sensitive information becomes inaccessible till: (i) the user presents their biometric features, or (ii) underlying cryptographic primitives are broken (e.g., AES, RSA, SHA and RSA), which is computationally infeasible. Even when all sensitive information is compromised, (i) the keys are not useable without the biometric data and (ii) the user can revoke them all and create a new set (i.e., reset compromised information). However, in the case of revocation, the application should define proper policies to handle previously signed or encrypted transactions, e.g., refreshing their signature and encryption.

Computed Information: Sensitive information such as biometric code ***K***, random key ***s***, private keys ***priv*** temporarily (i.e., for a very short time interval in the order of miliseconds) stays in the memory in cleartext form during CertainID computations. A resident malware may, in theory, be able to dump memory image at a correct time and access these credentials. This is a typical, but not practical, security risk which is valid for almost all applications in all operating systems dealing with sensitive data. Even when all sensitive information is extracted from the memory and compromised, the compromised CertainID information poses low risk and has no utility for the attacker as the user can revoke all of the credentials (i.e., biometric code ***K***, random key ***s***, private keys priv), create a new set of credentials and refresh the existing signatures and encryptions. As the additional layer of protection, appropriate firewall and intrusion protection combined with disk encryption should be considered as mandatory for the computer running the CertainID solution. Major processor vendors (e.g., Intel SGX) have developed CPU memory encryption capabilities to prevent such risks. In such computers, applications stay in an encrypted memory zone until they are fetched by CPU. Memory encryption and decryption are handled by CPU, and no operating system processes can access encryption keys. This technology is at its infancy currently and developing at a fast pace.

Software Vulnerabilities: CertainID and biometric sensor API's are developed as a proof-of-concept demonstrator. Although the engineering team have spent an extra effort to follow secure programming practices, bugs and vulnerabilities are inevitable, e.g., buffer overflow vulnerability which can be used to leak sensitive information. Although, such leaks pose very low risk and the leaked information has no utility for the attackers as explained in above paragraphs, it is a good practice to reduce the number and potential impacts of these vulnerabilities: (i) the applications developed on CertainID should pass through checks for secure programming principals and penetration tests regularly, (ii) third-party libraries (e.g., crypto library) and software components (e.g., database manager) used in the solution should be regularly monitored for their vulnerabilities, (iii) any patches to these third-party vulnerabilities should be investigated and applied with no delay.

3 CertainID Solution

3.1 System Design

In this section, we give details of the design and architecture of CertainID solution for school scenario where educational transactions are signed by CertainID.

3.1.1 Requirements Analysis

We collected requirements from both our clients and internal stakeholders to develop the demo system. All collected requirements are classified into functional requirements and non-functional requirements as follows.

- Functional requirements
  - The demo system shall run on MS Windows and consists of four applications; Server, Signer, Verifier and Monitor.
  - The Server shall provide application management functions (create, read, update, delete).
  - The Server shall provide student management functions (create, read, update, delete).
  - The Signer shall provide a signing function for any files.
  - The Signer shall write signatures and public keys in the same folder of original documents.
  - The Verifier shall authenticate documents with given signatures and public keys.
  - The Monitor shall display all logs generated since the Monitor started.
- Non-functional requirements
  - The system shall store important data securely.
  - The system shall be reasonably fast and accurate enough to demo (especially iris recognition).
  - The system shall be reliable enough to demo.

3.1.2 Use Case Diagram

As shown in Figure 7, we designed a use case diagram to meet the functional requirements. We retrieved three main actors, and details of these actors are given below:

- Administrator
  - Manages applications (courses) including creating, reading, updating and deleting.
  - Enrol students on applications.
  - Update students detail information.
  - Observe system logs.
- Teacher
  - Manages applications (courses) including creating, reading, updating and deleting.
  - Verifies documents which are submitted by students.
- Student

– Signs on documents to submit to teachers.

– Verifies his/her signed documents before submission.

– Provides his/her iris for enrolling on applications, updating his/her details or signing on documents.

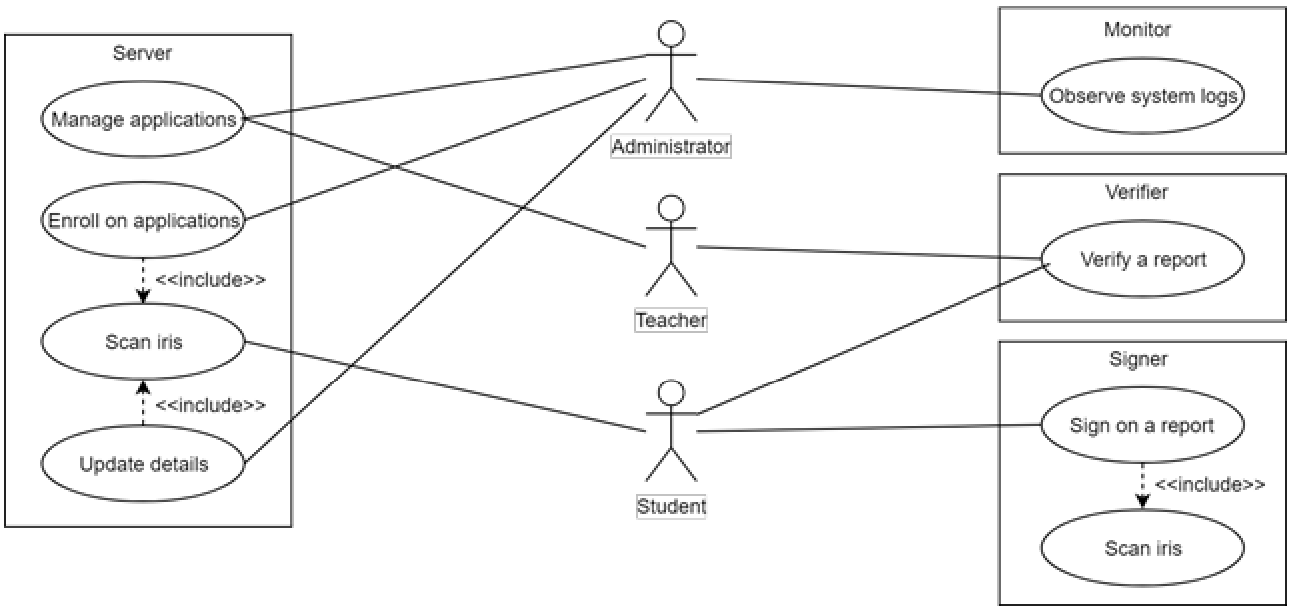


Figure 7 A use case diagram for the CertainID demo system.

3.1.3 Sequence Diagram

We developed a sequence diagram to show message flows clearly. Figure 8 shows message flows and relations between actors and objects (CertainID applications). This sequence diagram consists of three major blocks; enrolment, signing and verification. The followings are detail of message flows.

1. Enrolment

   a. An administrator reviews application list and modifies the list.
   b. The Server returns corresponding results to the administrator.
   c. A student comes to the Admin Office and request an enrolment to the administrator.
   d. The administrator fills up an enrolment form with the student’s detail information.
   e. The administrator asks the student to scan his/her iris.
   f. The student scans his/her iris with an iris scanner.
   g. The Server returns an iris scanning result to the student.
   h. The student informs the administrator that iris scanning has been completed.
   i. The Server returns an enrolment result to the administrator.

2. Signing

   a. The student searches his/her details from the Signer.
   b. The Signer displays the student’s detail information and applications enrolled.
   c. The student drags and drops a report file on the drop area in the Signer and selects an application he/she wants to submit for.

d. The Signer asks the student to scan his/her iris.
e. The student scans his/her iris with an iris scanner.
f. The Signer returns an iris scanning result to the student.
g. The Singer signs on the report file with a secret decrypted by the iris code. As a result, the Signer generates a signature file and a public key file in the same folder which the report file is in.

3. Verify the report

a. The student submits the signed report with the signature file and the public key file.
b. A teacher executes the Verifier to verify the report before scoring.
c. The teacher drags and drops the submitted report on the drop area in the Verifier.
d. The Verifier reads the signature file and the public key file.
e. The Verifier validates the public key with DB.
f. The Verifier verifies the signature and return a verification result.
g. Once the signature is authentic, the teacher marks score.
h. The teacher notifies the score to the student.

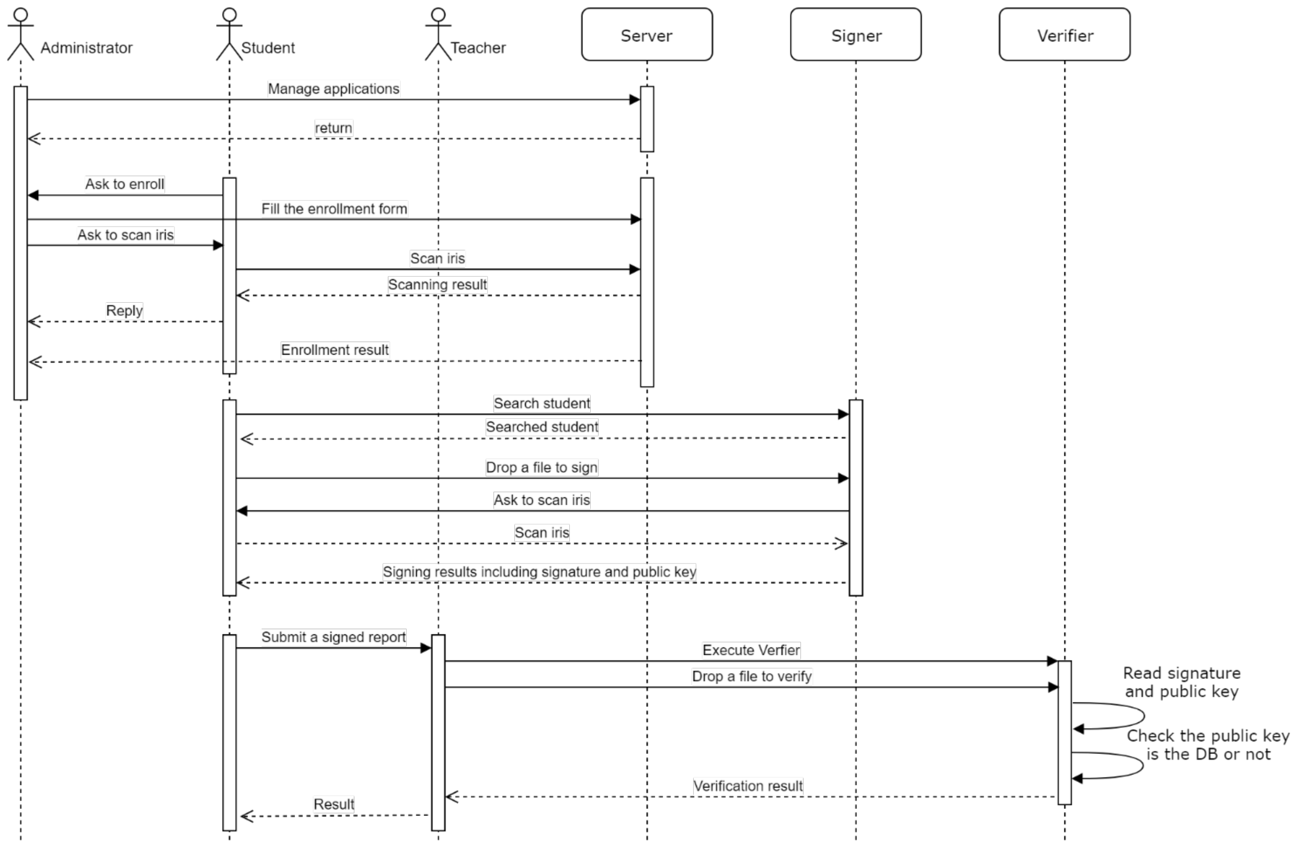


Figure 8 A sequence diagram for the CertainID demo system.

3.1.4 Class Diagram

Figure 9 shows a high-level class diagram for the CertainID demo system. This diagram shows only relations between classes. Figure 10 and Figure 11 show more detail information of classes which including members and methods.

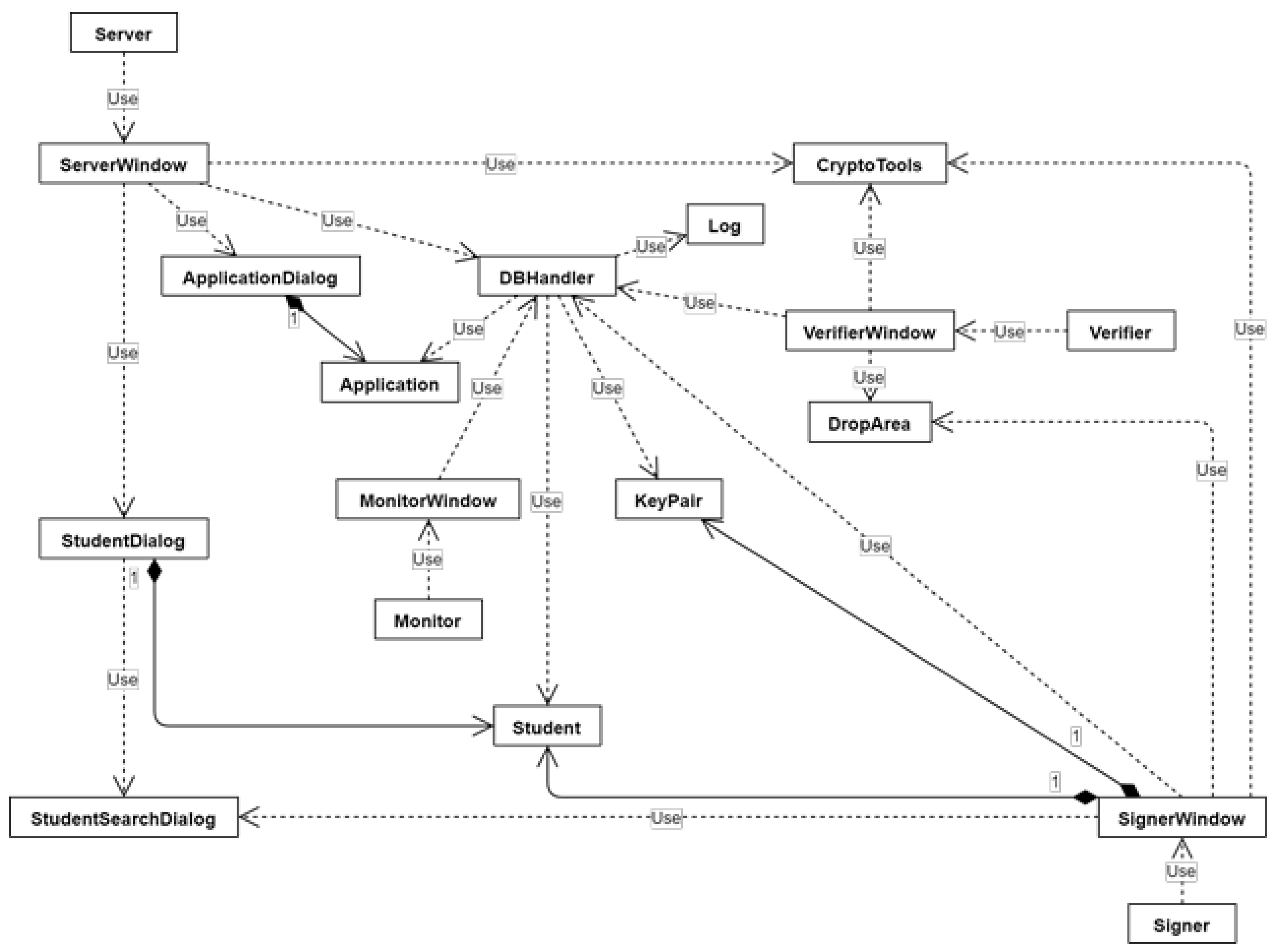


Figure 9 A high-level class diagram for the CertainID demo system.

In Figure 9, there are four main classes which represent applications:

- Server: This class represents the Server application and uses ServerWinodw as a main UI.
- Signer: This class represents the Signer application and uses SignerWindow as a main UI.
- Verifier: This class represents the Verifier application and uses VerifierWindow as a main UI.
- Monitor: This class represents the Monitor application and uses MonitorWindow as a main UI.

‘algorithms’ Package

Figure 10 shows a class diagram for ‘algorithms’ Package. This package is a group of algorithm-related classes, and details of these classes are given below:

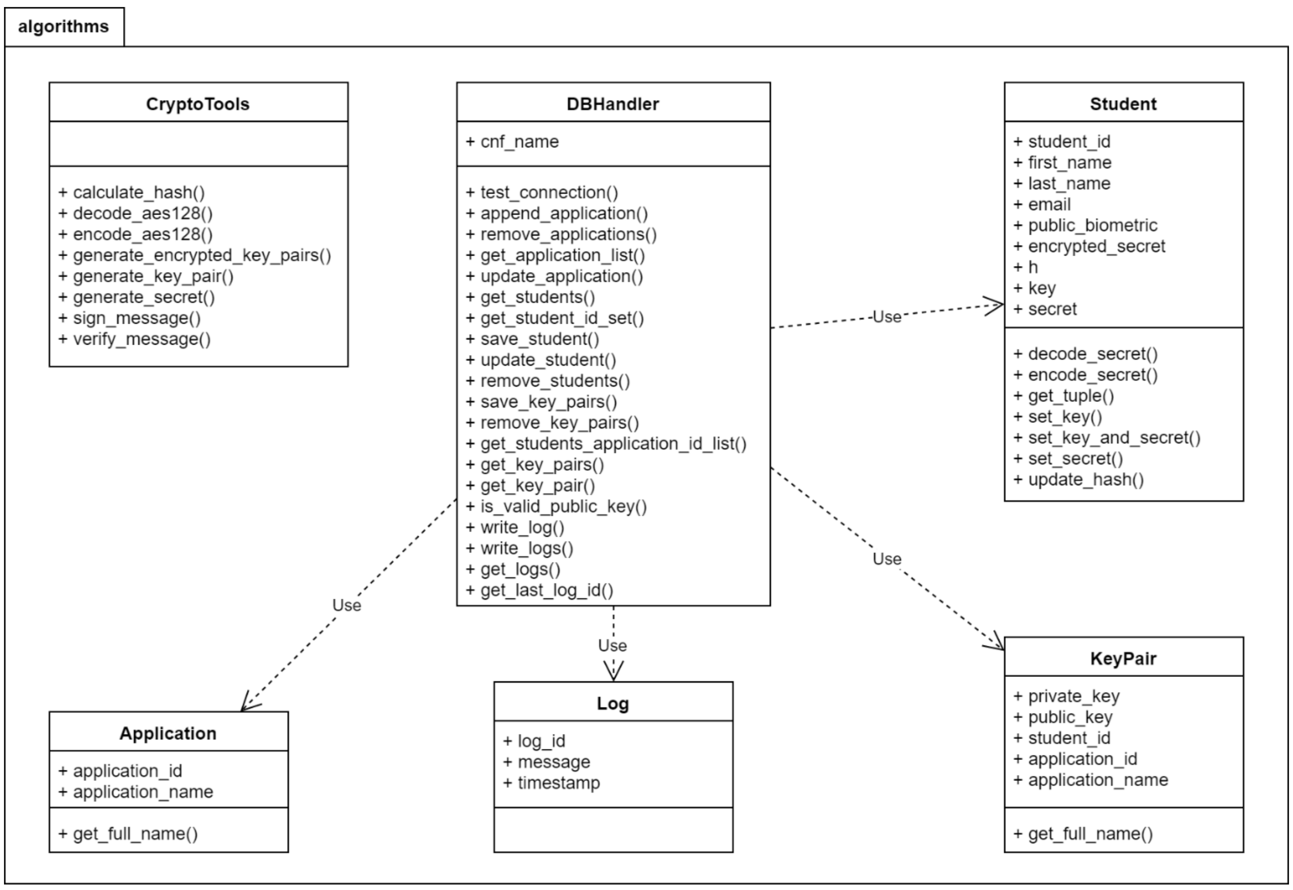


Figure 10 A class diagram for 'algorithms' package.

- CryptoTools: This class provides static crypto methods.
  - calculate_hash(): Calculates hash value of secret and student ID.
  - decode_aes128(): Decodes the given cipher text with AES128 algorithm.
  - encode_aes128(): Encodes the given message with AES128 algorithm.
  - generate_encrypted_key_pairs(): Generates given number of key pairs in which private keys will be encrypted with the secret.
  - generate_key_pair(): Generates a RSA key pair.
  - generate_secret(): Generates a secret code of given length.
  - sign_message(): Signs on given message and verify the signature with given public key.
  - verify_message(): Verifies signature with given public key.
- DBHandler: This class handles all database input/output operations.
  - cnf_name: DB configuration file’s name.
  - test_connection(): Tests the DB configurations.
  - append_application(): Adds applications to the DB. Application ID will be assigned automatically.
  - get_application_list(): Returns all application list in the application table.
  - remove_applications(): Deletes applications in the DB.
  - update_application(): Updates an application's name which matches with given application ID.
  - get_students(): Returns all students where student ID contains the query string.

  - get_student_id_set(): Returns a set of all students' ID.
  - get_students_application_id_list(): Returns a student's registered application list.
  - save_student(): Saves a student's information into the student table.
  - update_student(): Updates a student's information in the student table.
  - remove_students(): Removes students which matched with student id.
  - save_key_pairs(): Saves key pairs into the key table.
  - remove_key_pairs(): Removes key pairs which matched with all combinations of student ID and application ID.
  - get_key_pairs(): Returns a student's all key pairs.
  - get_key_pair(): Returns a key pair which matches with student ID and application ID.
  - is_valid_public_key(): Checks whether the key is in the DB or not.
  - get_last_log_id(): Returns last log ID.
  - get_logs(): Returns logs since given seconds ago.
  - write_log(): Writes message into the log table.
  - write_logs(): Writes message into the log table.
- Application: This class represents an application.
  - application_id: Application ID.
  - application_name: Application name.
- Student: This class represents a student.
  - student_id: Student ID.
  - first_name: Student’s first name.
  - last_name: Student’s last name.
  - email: Student’s email address.
  - public_biometric: Biometric of iris code.
  - encrypted_secret: Encrypted personal secret code.
  - h: Hash value of secret and student ID.
  - key: An iris code.
  - secret: For storing decrypted personal secret code.
- KeyPair: This class represent a pair of public key and private key.
  - private_key: A private key of an application.
  - public_key: A public key of an application.
- Log: This class represent a log information.
  - log_id: Log ID.
  - message: Log message.
  - timestamp: Timestamp of log.

‘ui’ Package

Figure 11 shows a class diagram for ‘ui’ Package. This package is a group of user interface classes, and details of these classes are given below:

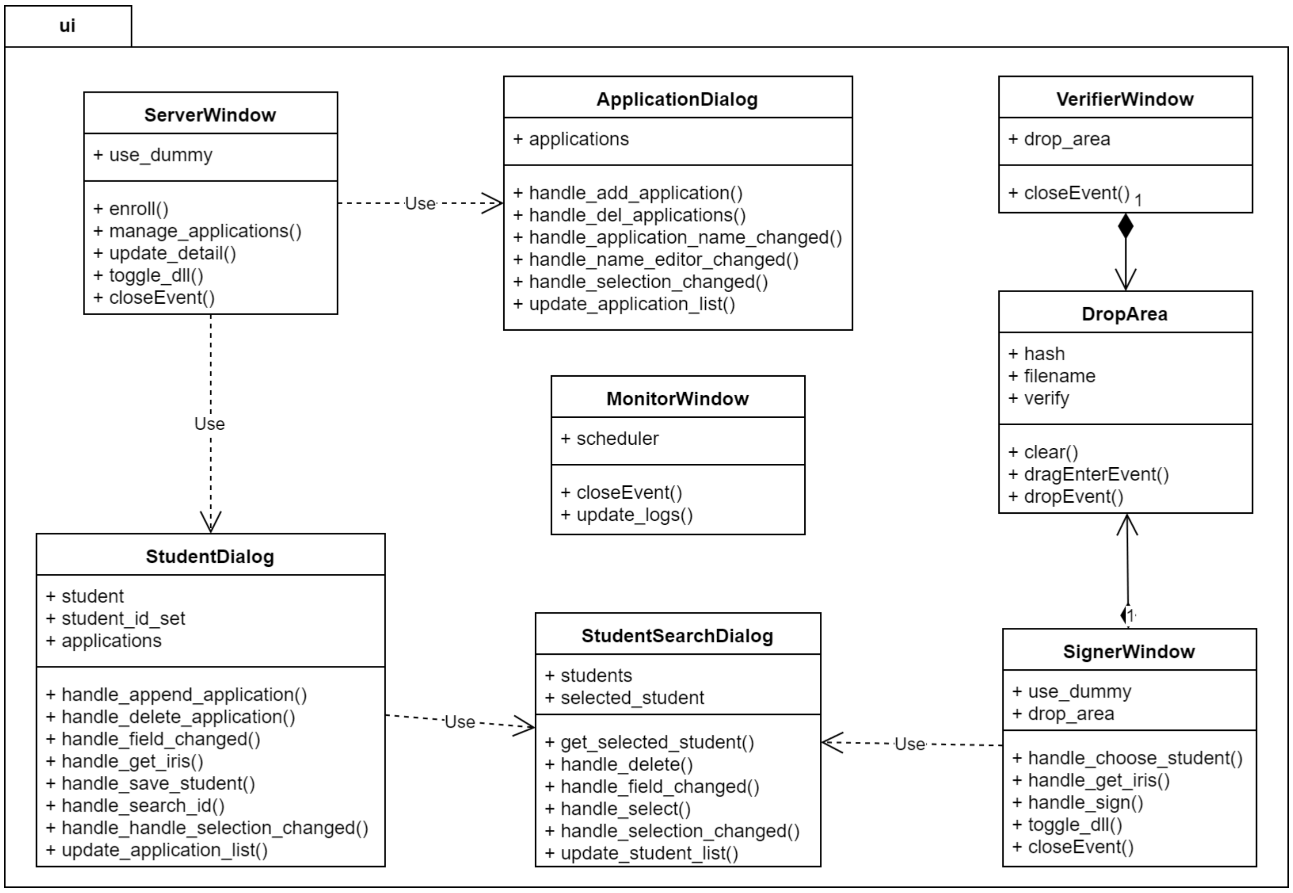


Figure 11 A class diagram for ‘ui’ package.

- ServerWindow: Main window UI for Server
  - use_dummy: Boolean for whether use a dummy DLL or not.
  - enroll(): Shows an instance of StudentDialog with ‘enroll’ mode.
  - manage_applications(): Shows an instance of ApplicationDialog.
  - update_detail(): Shows an instance of StudentDialog with update mode.
  - toggle_dll(): Toggles dummy DLL mode.
  - closeEvent(): Handles ‘closeEvent’.
- ApplicationDialog: Dialog for application management.
  - applications: List of Application objects.
  - handle_add_application(): Adds an application to list and DB.
  - handle_del_applications(): Deletes selected applications from list and DB.
  - handle_application_name_changed(): Update an application’s name.
  - handle_name_editor_changed(): Handles UI components when ‘edit_name’ field is changed.
  - handle_selection_changed(): Handles UI components when applications are selected.
  - update_application_list(): Refreshes application list.
- StudentDialog: Dialog for editing student information.

– student: Current student.

– student_id_set: All student ID list.

– applications: All application list.

– handle_append_application(): Handles application append actions.

– handle_delete_application(): Handles application removal actions.

– handle_field_changed(): Handles UI components if there're some fields changed.

– handle_get_iris(): Starts to scan iris.

– handle_save_student(): Saves current student information to DB.

– handle_search_id(): Shows an instance of StudentSearchDialog.

– handle_handle_selection_changed(): Handles selection changes in the application list.

– update_application_list(): Refreshes the application list.

- StudentSearchDialog: Dialog for searching students from DB.

  – students: Student list which matched with the query string.

  – selected_student: Currently selected student.

  – get_selected_student(): Returns currently selected student.

  – handle_delete(): Handles student removal actions.

  – handle_field_changed(): Handles UI components if there're some fields changed.

  – handle_select(): Handles 'Select' button clicks.

  – handle_selection_changed(): Handles selection changes in the student list.

  – update_student_list(): Refreshes the student list.

- SignerWindow: Main window UI for Signer.

  – use_dummy: Boolean for whether use a dummy DLL or not.

  – drop_area: An instance of DropArea with signing mode.

  – handle_choose_student(): Shows an instance of StudentSearchDialog.

  – handle_get_iris(): Starts to scan iris.

  – handle_sign(): Handles 'Sign' button click.

  – toggle_dll(): Toggles dummy DLL mode.

  – closeEvent(): Handles 'closeEvent'.

- DropArea: UI component to support drag and drop.

  – filename: Dropped filename.

  – hash: SHA256 of dropped file.

  – verify: DropArea mode. Either signing or verifying.

  – clear(): Clear all internal values.

  – dragEnterEvent(): Handles 'dragEnterEvent'.

  – dropEvent(): Handles 'dropEvent'.

- VerifierWindow: Main window UI for Verifier.

  – drop_area: An instance of DropArea with verifying mode.

  – closeEvent(): Handles 'closeEvent'.

- MonitorWindow: Main window UI for Monitor.
  - scheduler: A timer for pulling logs periodically.
  - update_logs(): Pulls logs and displays the logs.
  - closeEvent(): Handles 'closeEvent'

3.1.5 Database

We compared NoSQL with RDBMS to find out which database is more suitable for the CertainID demo system. Table 3 shows a summary of comparison between NoSQL and RDBMS. Nowadays NoSQL is very popular, and it supports non-structural data very well and more suitable for rapidly growing data. On the other hand, RDMBS is more suitable for structed data and traditional transaction systems. By considering the requirements carefully, we decided to use RDBMS as the CertainID is a traditional transaction system and the systems handles only small amount of data.

Table 3 Comparison between NoSQL and RDBMS.

CATEGORY	NOSQL	RDBMS
Use case	Real-time analytics, Content management, IoT, Mobile Apps	Legacy applications or applications that require multi-row transactions (i.e. accounting systems)
Data structure	No schema definition required	Structured data with clear schema
Risk	Less risk of attack due to design	Risk of SQL injection attacks
Analysis	A grate choice if you have unstructured and/or structured data with the potential for rapid growth	A great choice if you have structured data and need a traditional relational database

We designed a database scheme for the CertainID demo system as shown in Figure 12. The CertainID demo system only stores normal information and encrypted important information. Iris code is never stored in the database.

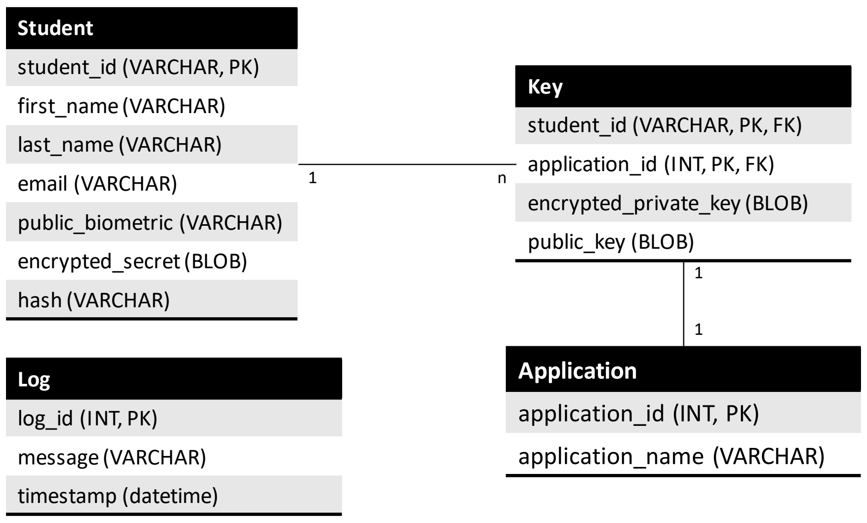


Figure 12 Database scheme for the CertainID.

The followings are detail information database tables.

- Student: Represents a student.
  - student_id: Student ID. Primary key of this table.
  - first_name: Student's first name.
  - last_name: Student's last name.
  - email: Student's email address.
  - public_biometric: Biometric of iris code.
  - encrypted_secret: Student's secret which encrypted by an iris code.
  - Hash: Hash value of student ID and student's secret.
- Application: Represents an application.
  - application_id: Application ID. Primary key of this table.
  - application_name: Application's name.
- Key: Represents a pair of private key and public key.
  - student_id: Student ID. A part of primary key. Foreign key for the Student table.
  - application_id: Application ID. A part of primary key. Foreign key for the Application table.
  - encrypted_private_key: Individual private key for an application which encrypted by a student's secret.
  - public_key: Individual public key for an application.
- Log: Represents a log record.
  - log_id: Log ID. Primary key of this table.
  - message: Log message.
  - timestamp: Timestamp for this record.

3.2 Implementation

3.2.1 Development Platform

The CertainID uses many crypto functions; AES128, SHA256, RSA2048 and PS256. To find out which crypto library is most suitable for our case, we compared some popular crypto libraries written in different programming languages as

Table 4. All of them supports all required crypto functions, and according to the licenses there's almost no restrictions to use[1].

[1] From January 2019, Oracle started to charge license fees on some Java products. To avoid license fees, users need to switch to OpenJDK.

Table 4 Comparison of crypto libraries.

CATEGORY	C++	JAVA	PYTHON
Library	Crypto++ Library 8.0	java.security	PyCryptodome 3.7.2
AES128	Yes	Yes	Yes
SHA256	Yes	Yes	Yes
RSA2048	Yes	Yes	Yes
PS256	Yes (RSASS)	Yes (JWSAlgorithm)	Yes (PKCS1_PSS)
License	<www.cryptopp.com/License.txt>	<www.java.com/en/download/faq/distribution.xml>	<pycryptodome.readthedocs.io/en/latest/src/license.html>

The CertainID should run on MS Windows environments. To meet this requirement, we listed up some candidates as shown in Table 5. As C++, Java and Python can meet all functional requirements and non-functional requirements, any of them would be a good development platform. We gave higher priorities on productivity and debugging, and finally we chose Python as a development platform to shorten development time.

Table 5 Comparison of development platforms.

CATEGORY	C++	JAVA	PYTHON
IDE	Visual Studio	Eclipse	PyCharm, Eclipse
GUI	MFC, Qt	JavaFX	PyQt
Executable file	Yes	Yes (JSmooth, exe4j, Launch4j)	Yes (py2exe, pyinstaller)
Performance	Very fast	Fast enough	Slow
Productivity	Medium	Good	Excellent
Debugging/ Maintenance	Medium (Smart pointer)	Easy (Garbage collection)	Easy (Garbage collection)

3.2.2 APIs for InfinityOptics Iris Scanner

The CertainID works with an iris scanner which developed by InfinityOptics. To obtain iris codes properly from the InfinityOptics iris scanner, we had to define APIs between the CertainID and InfinityOptics SDK which is written C. Requirements for APIs are given below:

- Registration API: Returns a 128-bit iris code, a 32-bit public biometric and an error code
- Authentication API: Caller provides a 32-bit public biometric. Returns a 128-bit iris code with an error code
- APIs can be in any form (DLL, EXE, and so on) if Python can call
- Platform: Windows 10 64bit

API Definitions

Figure 13 shows agreed APIs between the CertainID and InfinityOptics SDK. There are two APIs; one for registration and the other one for authentication.

```
#ifdef DLLTEST_EXPORTS

#define EXPORT extern "C" __declspec(dllexport)

#else

#define EXPORT extern "C" __declspec(dllimport)

#endif


// return: 0 – No error, Other values – Error code

EXPORT int _stdcall infinity_lens_plus_biometric_register (char *pCode, char *pMask);

EXPORT int _stdcall infinity_lens_plus_biometric_authenticate (const char * pMask, char * pCode);
```

Figure 13 APIs for InfinityOptics.

- Registration: infinity_lens_plus_biometric_register()
  - Parameters
    - pCode: A char array for iris code. It must be allocated by caller, and size of array must be bigger than 32 bytes.
    - pMask: A char array for biometric. It must be allocated by caller, and size of array must be bigger than 9 bytes.
  - Returns
    - Error code: Returns 0 if there's no error. Otherwise returns an error code defined.
- Authentication: infinity_lens_plus_biometric_authenticate()
  - Parameters
    - pCode: A char array for iris code. It must be allocated by caller, and size of array must be bigger than 32 bytes.
    - pMask: A constant char array for biometric. Caller must provide a valid biometric to get a proper iris code.
  - Returns
    - Error code: Returns 0 if there's no error. Otherwise returns an error code defined.

API Usages in Python

Figure 14 shows how to use the APIs in Python. This example does not handle any exception or error codes of APIs.

```
import ctypes as c
dll = c.CDLL("InfinityLensPlusCore.dll")

# registration
reg_func = dll['infinity_lens_plus_biometric_register']
reg_func.restype = c.c_int
reg_func.argtypes = (c.c_char_p, c.c_char_p)
biometric = c.create_string_buffer(9)
iris_code = c.create_string_buffer(33)
reg_result = reg_func(new_iris_code, biometric)

# authentication
auth_func = dll['infinity_lens_plus_biometric_authenticate']
auth_func.restype = c.c_int
auth_func.argtypes = (c.c_char_p, c.c_char_p)
new_iris_code = c.create_string_buffer(33)
copied_biometric = biometric
auth_result = auth_func(new_iris_code, copied_biomeric)
```

Figure 14 A usage example of the APIs in Python.

3.3 Installation

3.3.1 Runtime Environment

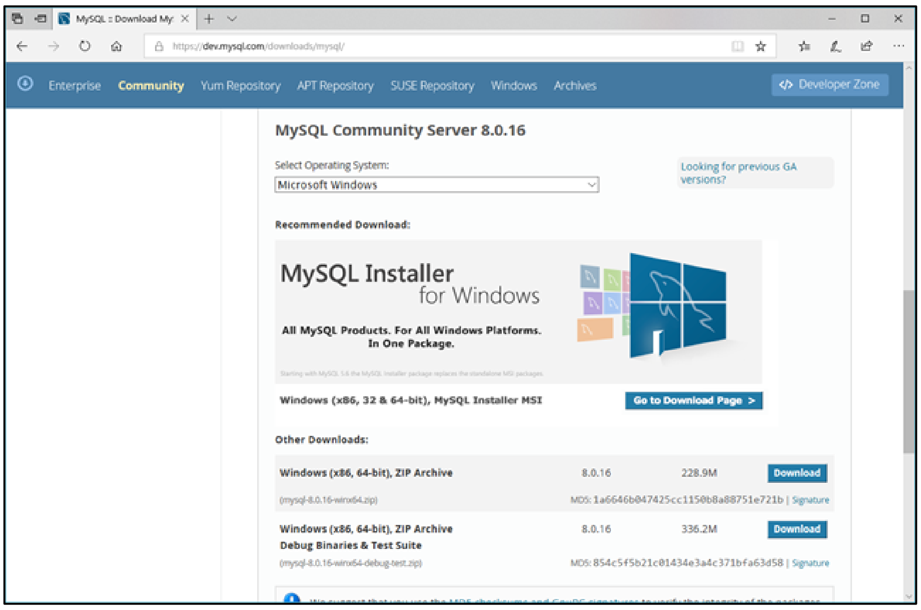
The CertainID runs on MS Windows 10 64-bit edition, and the followings are more detail about the running environment.

- OS: MS Windows 10 64-bit edition.
  - Does not support 32-bit edition.
- CPU/RAM: No specific requirements.
  - Just enough if a PC can run on MS Windows 10 64-bit.
  - The faster, the better to boot iris recognition speed.
- DBMS: MySQL Community Server 8.0 64-bit edition.
  - The latest version is recommended.

3.3.2 Install MySQL Server

Before install the CertainID, setup a MySQL server first. We recommend installing MySQL Community Server 8.0 or higher version as it's more secure. If you have a MySQL server, you can skip this part. Otherwise, please follow the instruction given:

1. Access <https://dev.mysql.com/downloads/windows/installer/8.0.html>, and download the latest version of MySQL



MySQL Community Server 8.0.16

Select Operating System:
Microsoft Windows

Recommended Download:

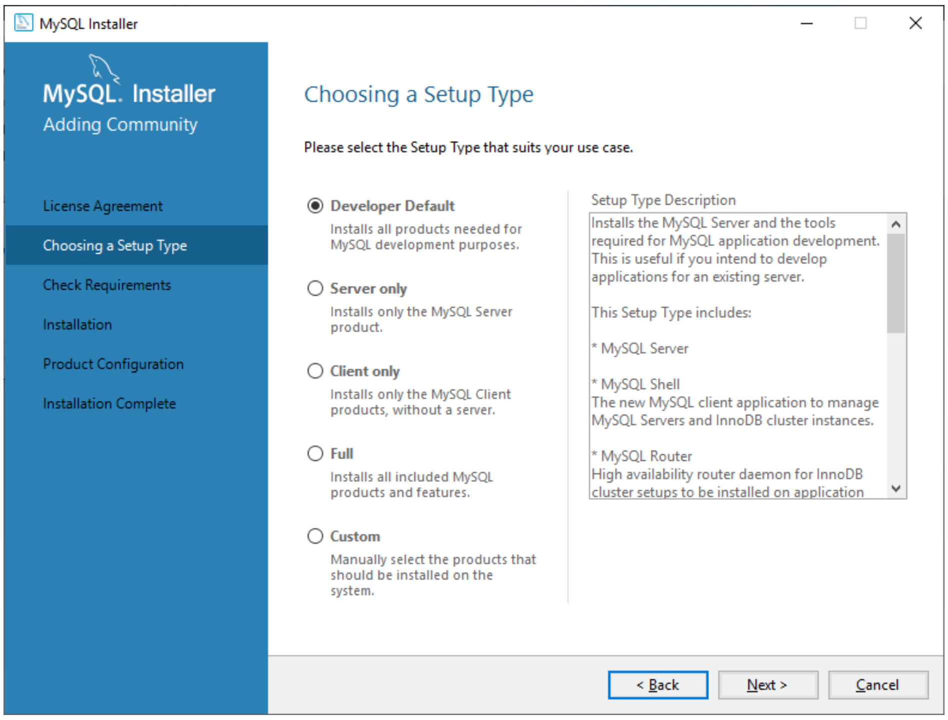
Windows (x86, 32 & 64-bit), MySQL Installer MSI

Other Downloads:

Windows (x86, 64-bit), ZIP Archive (mysql-8.0.16-winx64.zip)	8.0.16	228.9M	MD5: 1a6646b047425cc1150b8a88751e721b
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite (mysql-8.0.16-winx64-debug-test.zip)	8.0.16	336.2M	MD5: 854c5f5b21c01434e3a4c371bfa63d58

Figure 15 MySQL Community Server download page.

2. Execute the downloaded file. For minimal install, select 'Custom' to select the MySQL product manually. Then, as shown in Figure 17, select MySQL Server 8.0 – X64 and MySQL Workbench 8.0 – X64. Or simply select either 'Developer Default' or 'Full' as shown in Figure 16. Please note that 'Developer Default' and 'Full' install may require to install Python 3.7 manually.


MySQL Installer

MySQL. Installer
Adding Community

License Agreement
Choosing a Setup Type
Check Requirements
Installation
Product Configuration
Installation Complete

Choosing a Setup Type

Please select the Setup Type that suits your use case.

[● Developer Default] Installs all products needed for MySQL development purposes.

○ Server only: Installs only the MySQL Server product.

○ Client only: Installs only the MySQL Client products, without a server.

○ Full: Installs all included MySQL products and features.

○ Custom: Manually select the products that should be installed on the system.

Setup Type Description

Installs the MySQL Server and the tools required for MySQL application development. This is useful if you intend to develop applications for an existing server.

This Setup Type includes:

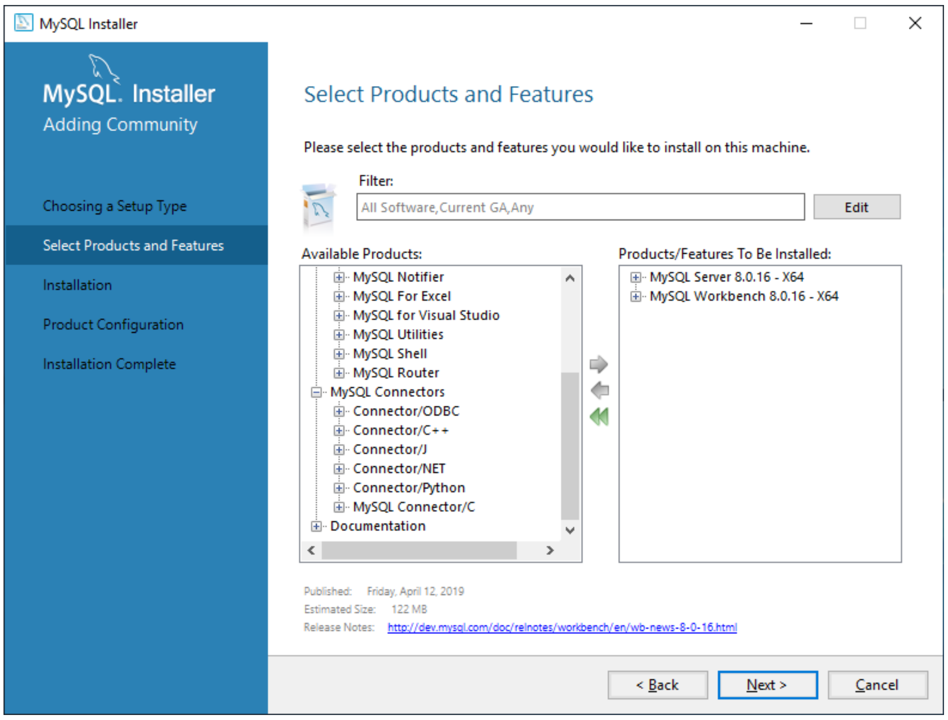
* MySQL Server

* MySQL Shell
The new MySQL client application to manage MySQL Servers and InnoDB cluster instances.

* MySQL Router
High availability router daemon for InnoDB cluster setups to be installed on application

< Back | Next > | Cancel

Figure 16 MySQL setup - Choosing a Setup Type.


MySQL Installer

MySQL. Installer
Adding Community

Choosing a Setup Type
Select Products and Features
Installation
Product Configuration
Installation Complete

Select Products and Features

Please select the products and features you would like to install on this machine.

Filter: All Software,Current GA,Any | Edit

Available Products:
- MySQL Notifier
- MySQL For Excel
- MySQL for Visual Studio
- MySQL Utilities
- MySQL Shell
- MySQL Router
- MySQL Connectors
  - Connector/ODBC
  - Connector/C++
  - Connector/J
  - Connector/NET
  - Connector/Python
  - MySQL Connector/C
- Documentation

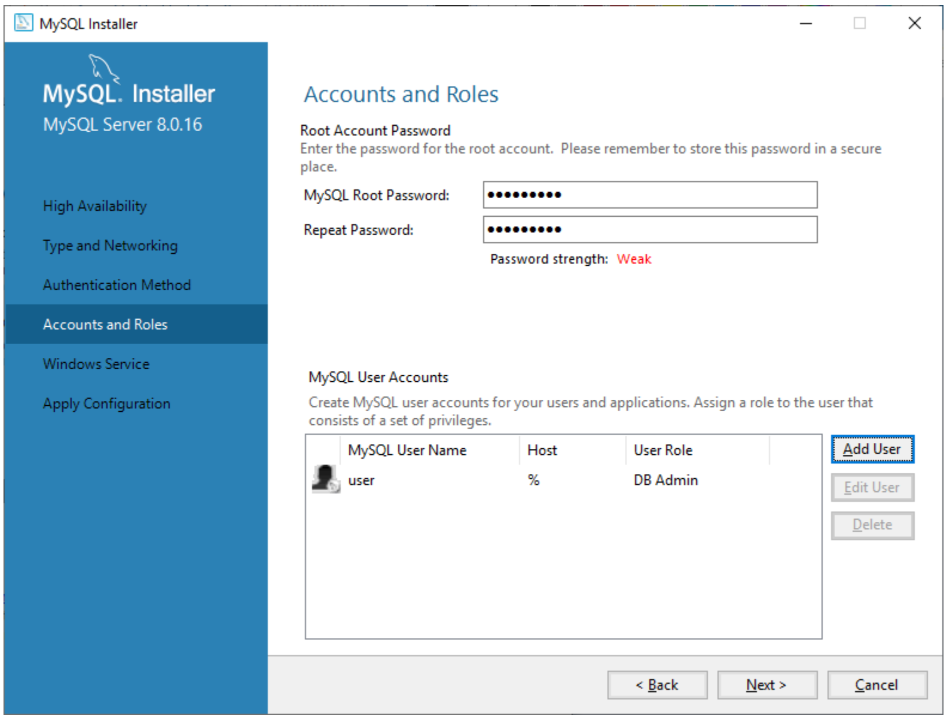
Products/Features To Be Installed:
- MySQL Server 8.0.16 - X64
- MySQL Workbench 8.0.16 - X64

Published: Friday, April 12, 2019
Estimated Size: 122 MB
Release Notes: http://dev.mysql.com/doc/relnotes/workbench/en/wb-news-8-0-16.html

< Back | Next > | Cancel

Figure 17 MySQL setup - Select Product and Features.

3. To setup MySQL, default values are enough. In the middle of setup, enter root password twice. We also recommend adding a user for accessing DB not as a root. To add a user, click the ‘Add User’ button, and fill up the fields.


MySQL Installer

MySQL. Installer
MySQL Server 8.0.16

Accounts and Roles

Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

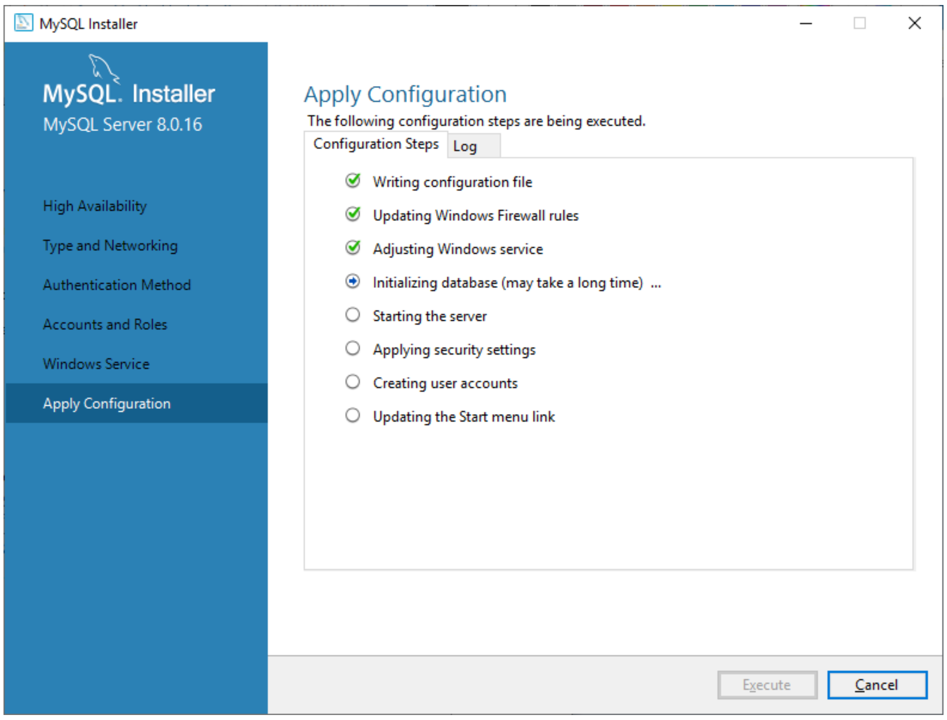
MySQL Root Password: ●●●●●●●●●
Repeat Password: ●●●●●●●●●
Password strength: Weak

MySQL User Accounts
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role
user	%	DB Admin

Figure 18 MySQL setup - Accounts and Roles.

4. After setup all values, the setup program will apply configuration.


MySQL Installer

MySQL. Installer
MySQL Server 8.0.16

Apply Configuration
The following configuration steps are being executed.

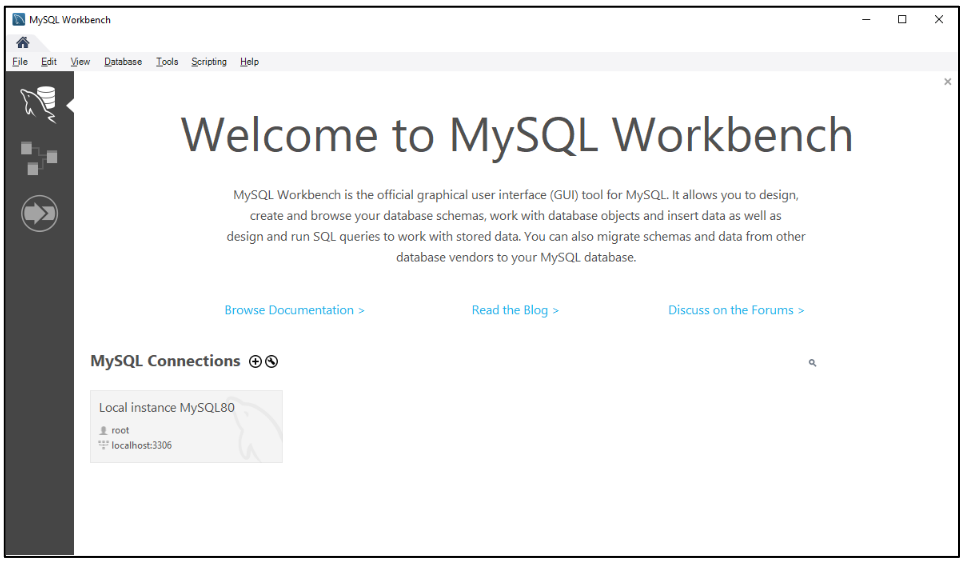
- ✓ Writing configuration file
- ✓ Updating Windows Firewall rules
- ✓ Adjusting Windows service
- ● Initializing database (may take a long time) ...
- ○ Starting the server
- ○ Applying security settings
- ○ Creating user accounts
- ○ Updating the Start menu link

Figure 19 MySQL setup - Apply Configuration.

3.3.3 Setup a DB Schema

MySQL provides a GUI tool, MySQL Workbench, to manage MySQL server easily. In this instruction, we use MySQL Workbench.

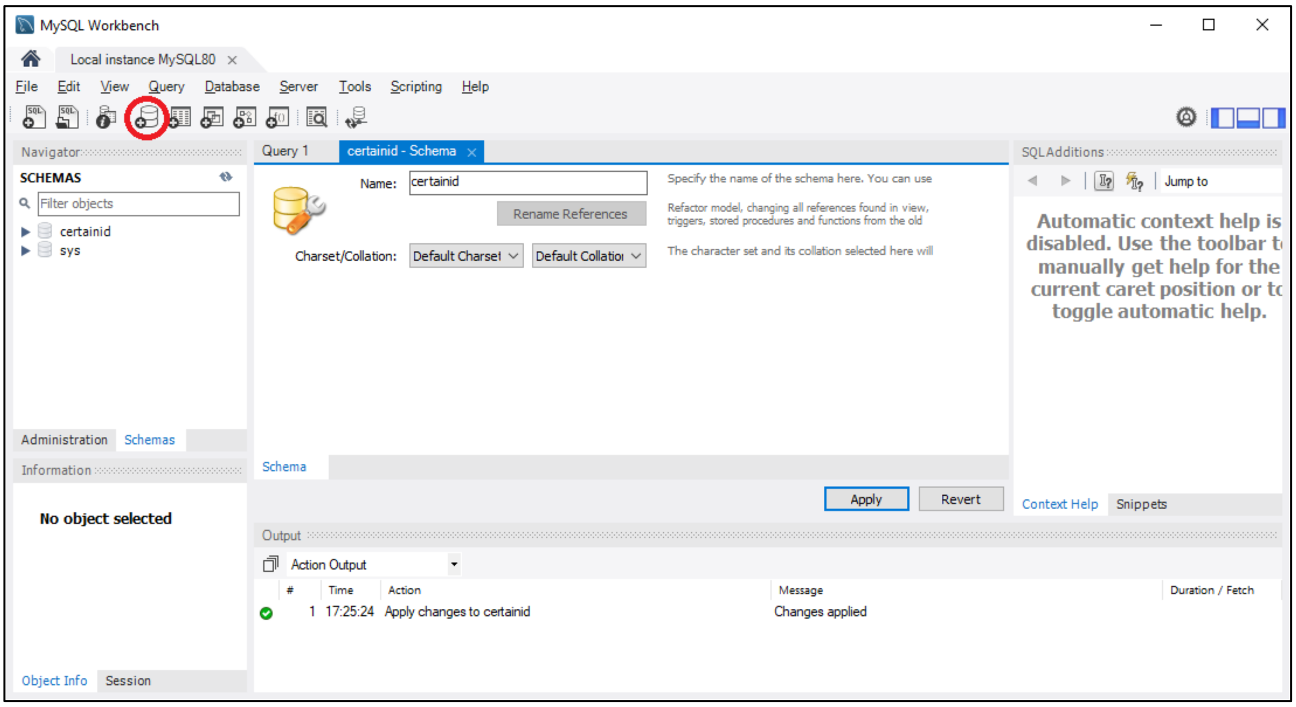
5. Execute MySQL Workbench, and access to the MySQL Server. If you use other MySQL servers, you can setup a new connection by clicking the circled plus icon as shown in Figure 20.



MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Figure 20 MySQL Workbench – Welcome screen.

6. To create a DB Schema, click 'Create a new schema' button (red circle in Figure 21). Then name 'certainid'.



Name: certainid

Charset/Collation: Default Charset | Default Collation

Specify the name of the schema here. You can use

Refactor model, changing all references found in view, triggers, stored procedures and functions from the old

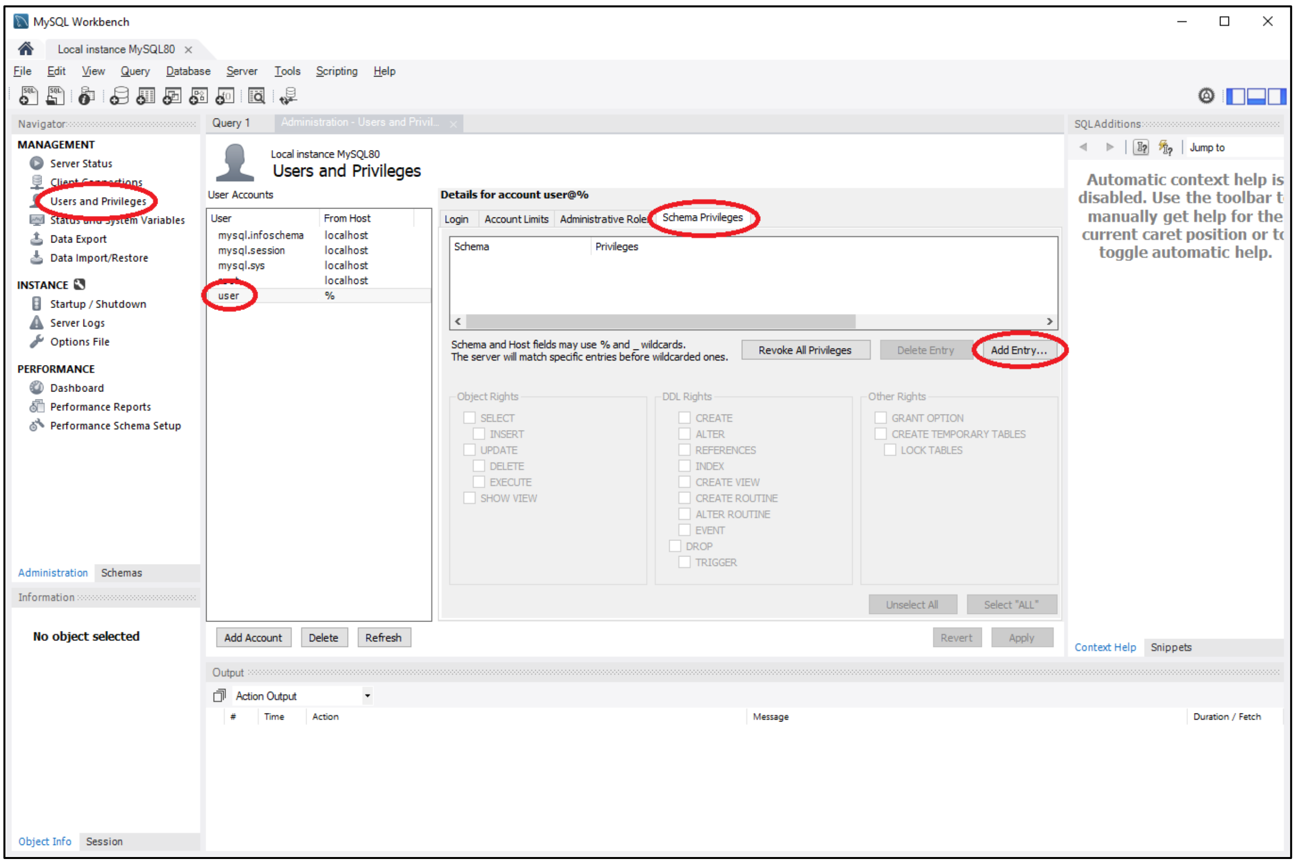
The character set and its collation selected here will

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

	#	Time	Action	Message	Duration / Fetch
✓	1	17:25:24	Apply changes to certainid	Changes applied	

Figure 21 Create a Schema.

7. To assign a user to the schema, click the 'User and Privileges' from the Navigator (please refer Figure 22). Then select a user wish to assign to the schema, select 'Schema Privileges' and click 'Add Entry' button.



User	From Host
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost
user	%

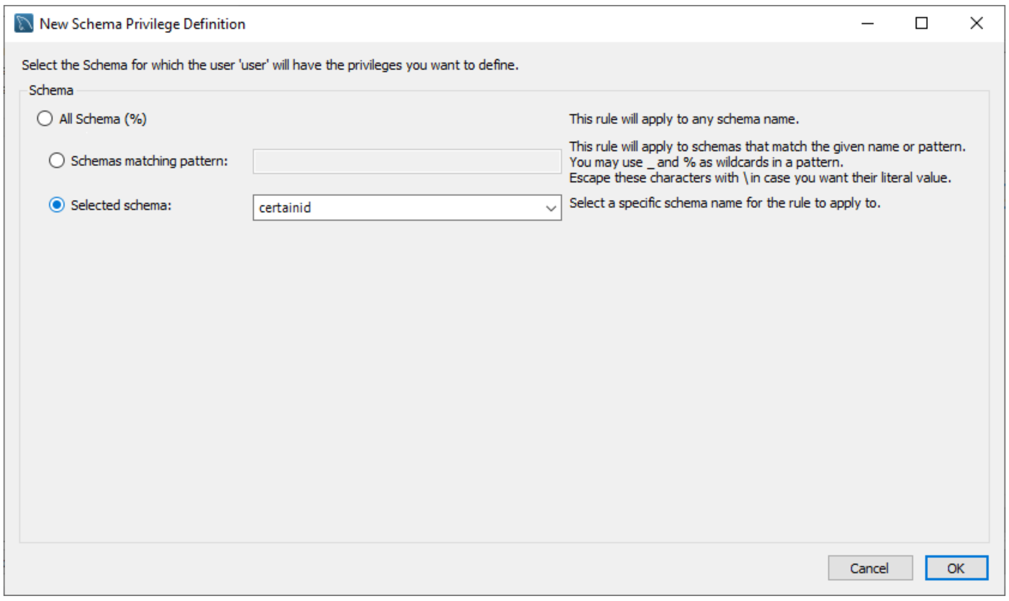
Schema and Host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.

No object selected

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or to toggle automatic help.

Figure 22 Edit Privileges

8. Select ‘certainid’ to assign rights.

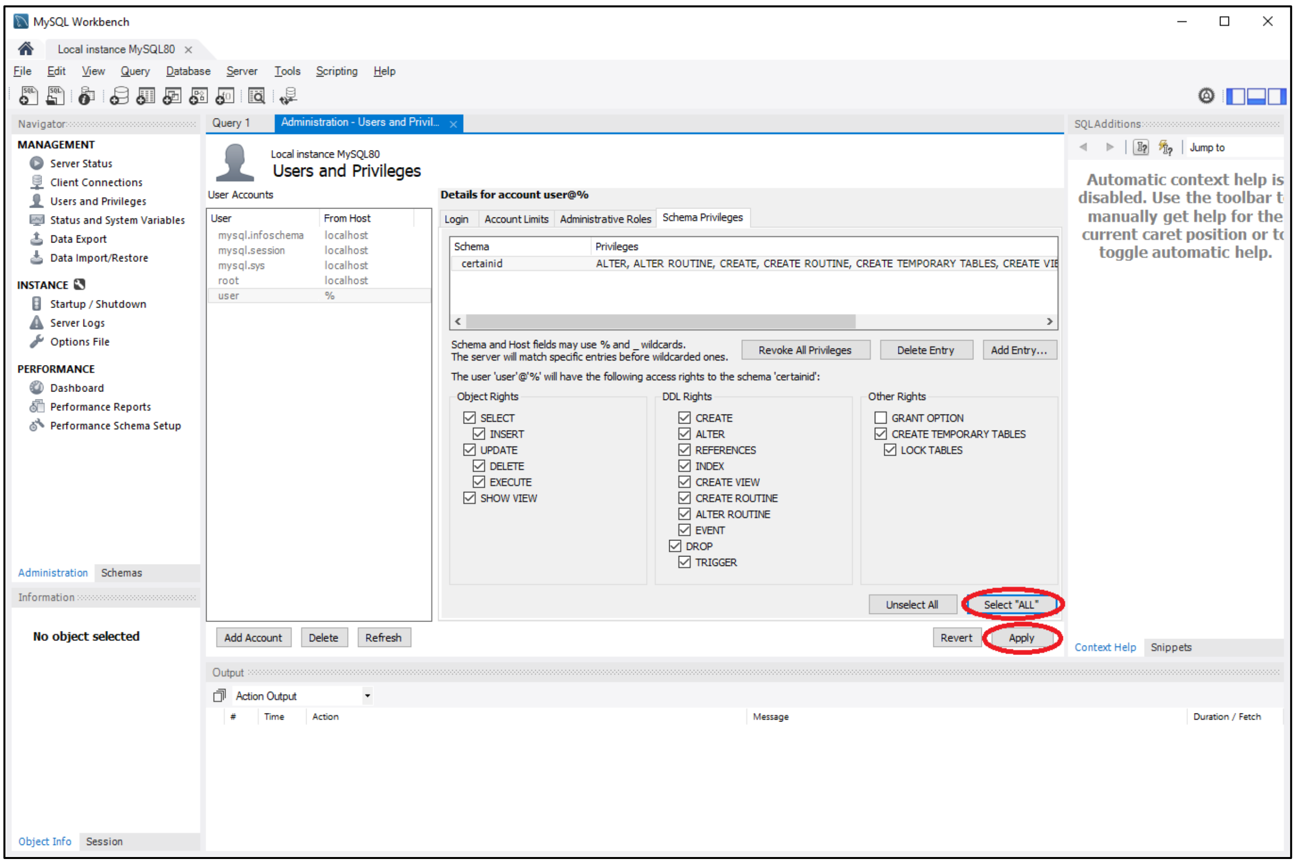


Select the Schema for which the user 'user' will have the privileges you want to define.

All Schema (%)	This rule will apply to any schema name.
Schemas matching pattern:	This rule will apply to schemas that match the given name or pattern. You may use _ and % as wildcards in a pattern. Escape these characters with \ in case you want their literal value.
[✓ Selected schema:] certainid	Select a specific schema name for the rule to apply to.

Figure 23 Select schema

9. Click ‘Select All’ button to assign all rights to the user. If necessary, add or remove rights. Then click ‘Apply’ to apply.



Local instance MySQL80
Users and Privileges

User Accounts

User	From Host
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost
user	%

Details for account user@%

Schema	Privileges
certainid	ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE VIE

Schema and Host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.

The user 'user'@'%' will have the following access rights to the schema 'certainid':

Object Rights
- ☑ SELECT
- ☑ INSERT
- ☑ UPDATE
- ☑ DELETE
- ☑ EXECUTE
- ☑ SHOW VIEW

DDL Rights
- ☑ CREATE
- ☑ ALTER
- ☑ REFERENCES
- ☑ INDEX
- ☑ CREATE VIEW
- ☑ CREATE ROUTINE
- ☑ ALTER ROUTINE
- ☑ EVENT
- ☑ DROP
- ☑ TRIGGER

Other Rights
- ☐ GRANT OPTION
- ☑ CREATE TEMPORARY TABLES
- ☑ LOCK TABLES

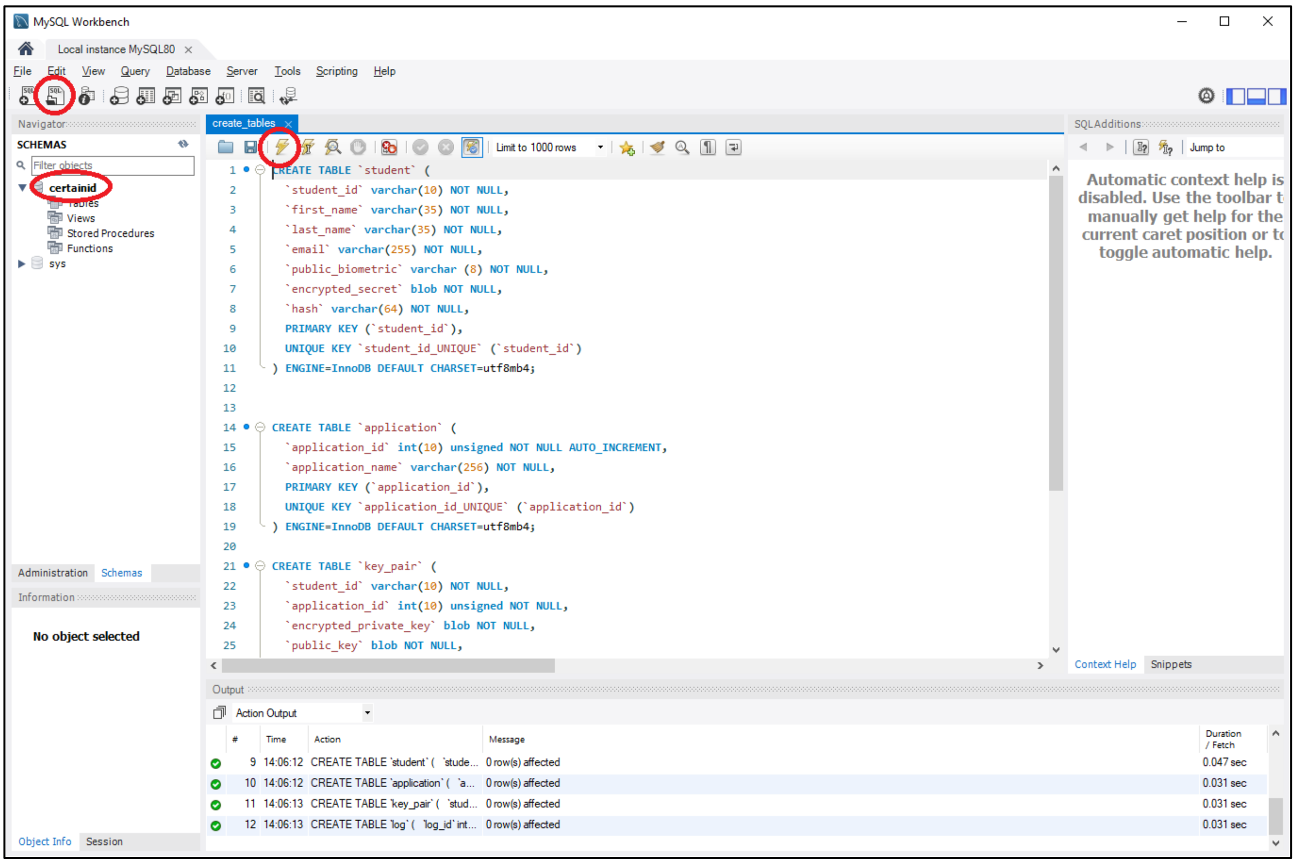
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or tc toggle automatic help.

No object selected

#	Time	Action	Message	Duration / Fetch

Figure 24 Append rights to the schema

10. To use ‘certainid’ schema, double click the ‘certainid’. Then ‘certainid’ schema will be in bold type (please refer Figure 25). Open ‘create_tables.sql’ file (distributed with the CertainID package) with the second tool button in the toolbar. Execute the loaded SQL statements with the lightening button in the sub-window toolbar.



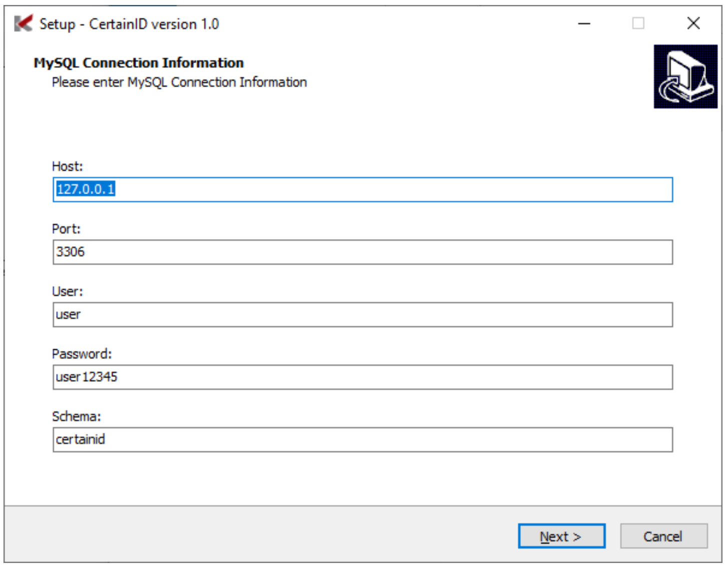
```
CREATE TABLE `student` (
  `student_id` varchar(10) NOT NULL,
  `first_name` varchar(35) NOT NULL,
  `last_name` varchar(35) NOT NULL,
  `email` varchar(255) NOT NULL,
  `public_biometric` varchar (8) NOT NULL,
  `encrypted_secret` blob NOT NULL,
  `hash` varchar(64) NOT NULL,
  PRIMARY KEY (`student_id`),
  UNIQUE KEY `student_id_UNIQUE` (`student_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;


CREATE TABLE `application` (
  `application_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `application_name` varchar(256) NOT NULL,
  PRIMARY KEY (`application_id`),
  UNIQUE KEY `application_id_UNIQUE` (`application_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `key_pair` (
  `student_id` varchar(10) NOT NULL,
  `application_id` int(10) unsigned NOT NULL,
  `encrypted_private_key` blob NOT NULL,
  `public_key` blob NOT NULL,
```

```
#   Time      Action                                  Message              Duration / Fetch
9   14:06:12  CREATE TABLE 'student' (  'stude...     0 row(s) affected    0.047 sec
10  14:06:12  CREATE TABLE 'application' (  'a...     0 row(s) affected    0.031 sec
11  14:06:13  CREATE TABLE 'key_pair' (  'stud...     0 row(s) affected    0.031 sec
12  14:06:13  CREATE TABLE 'log' (  'log_id' int...   0 row(s) affected    0.031 sec
```

Figure 25 Create tables.

3.3.4 Install CertainID

11. To install the CertainID, execute “CertainID.exe”. Figure 26 shows the first page, MySQL Connection Information, of the CertainID installer. All fields are filled with default values. If you used different values for setting up the DB, fill the values you used.

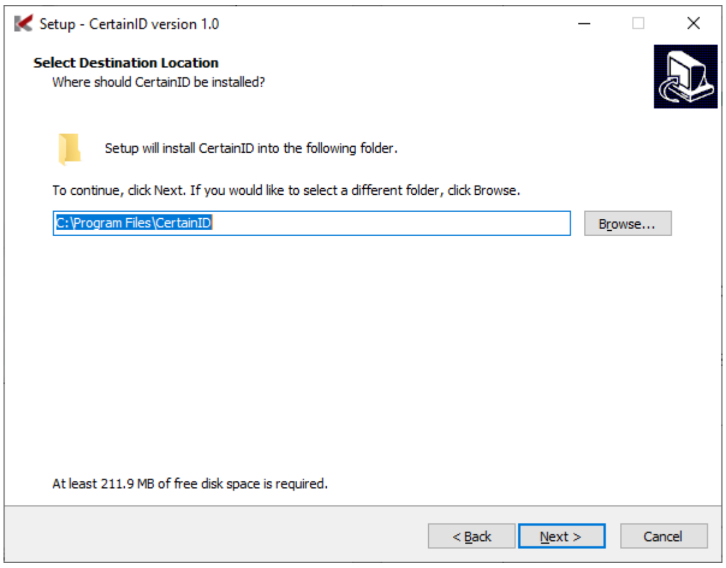


MySQL Connection Information
Please enter MySQL Connection Information

Host:	127.0.0.1
Port:	3306
User:	user
Password:	user12345
Schema:	certainid

Figure 26 Setup CertainID - MySQL Connection Information.

12. Select destination location to install the CertainID. By default, the CertainID will be installed in ‘C:\Program Files\CertainID’.



Select Destination Location

Where should CertainID be installed?

Setup will install CertainID into the following folder.

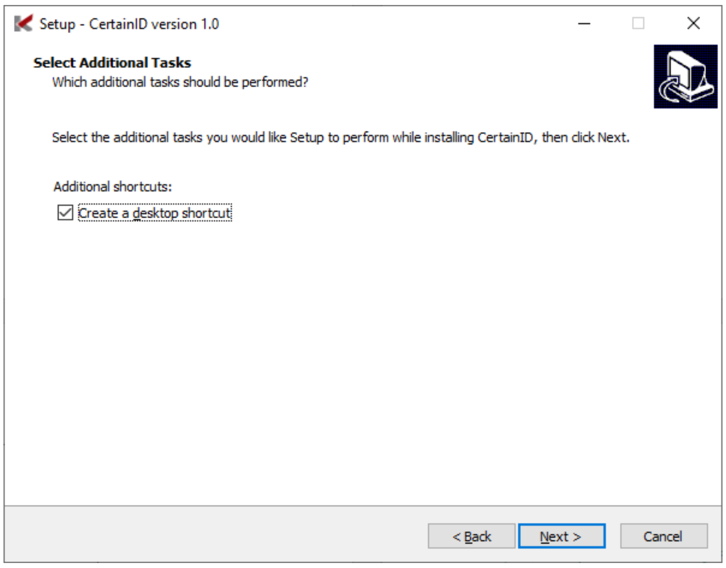
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\CertainID

At least 211.9 MB of free disk space is required.

Figure 27 Setup CertainID - Destination Location.

13. Choose whether create desktop shortcuts or not.



Select Additional Tasks

Which additional tasks should be performed?

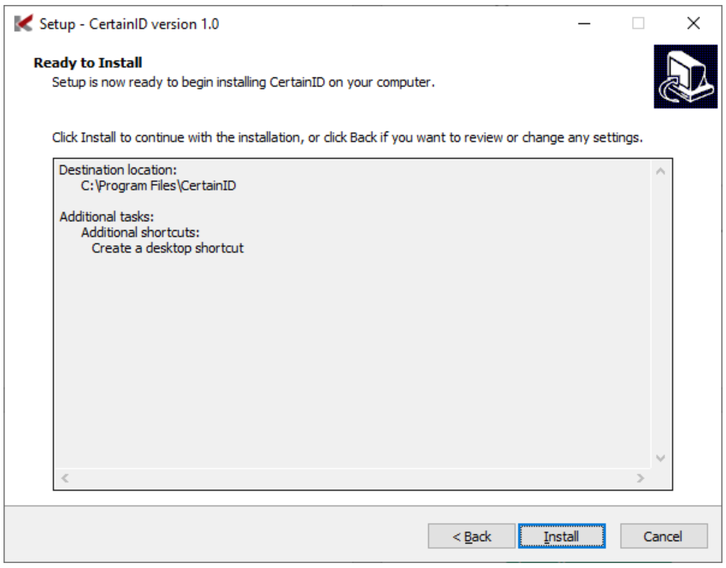
Select the additional tasks you would like Setup to perform while installing CertainID, then click Next.

Additional shortcuts:

☑ Create a desktop shortcut

Figure 28 Setup CertainID - Create desktop shortcuts.

14. Review your selections. Click ‘Back’ if you want to change any settings. Or click ‘Install’ to install the CertainID.


Setup - CertainID version 1.0

Ready to Install

Setup is now ready to begin installing CertainID on your computer.

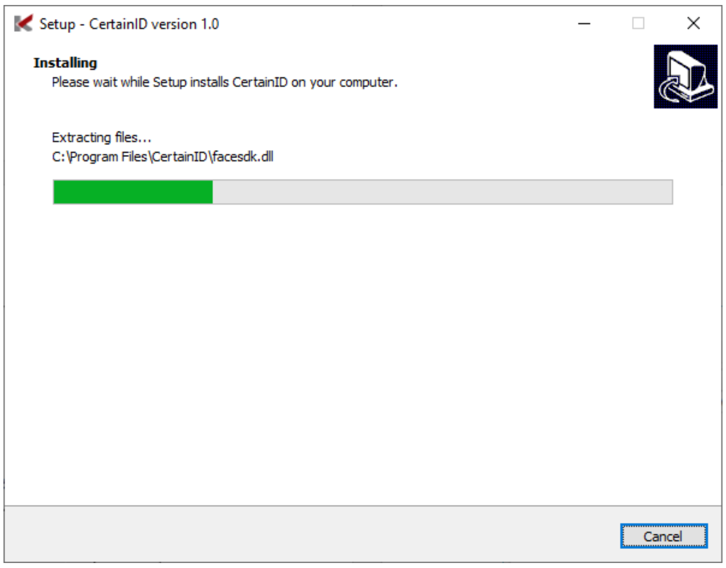
Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination location:
C:\Program Files\CertainID

Additional tasks:
Additional shortcuts:
Create a desktop shortcut

Figure 29 Setup CertainID - Review configurations.

15. Figure 30 shows progress of installation.


Setup - CertainID version 1.0

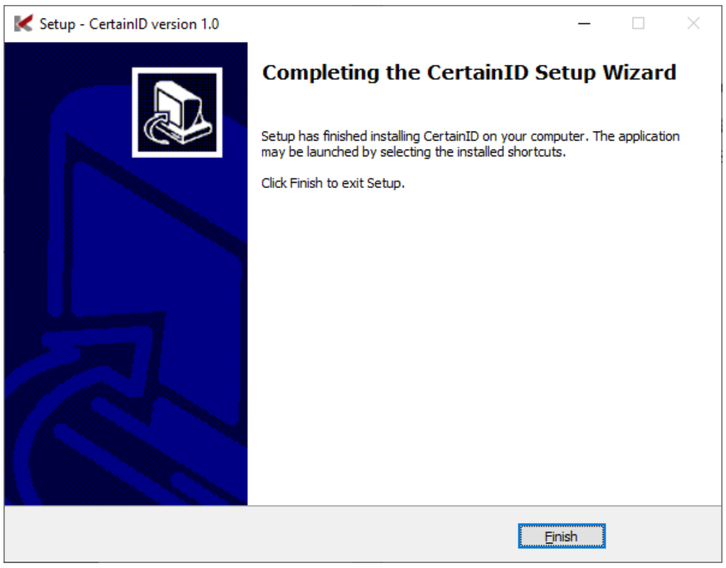
Installing

Please wait while Setup installs CertainID on your computer.

Extracting files...
C:\Program Files\CertainID\facesdk.dll

Figure 30 Setup CertainID – Installing

16. Once installation is finished, click 'Finish' to exit the setup program.


Setup - CertainID version 1.0

Completing the CertainID Setup Wizard

Setup has finished installing CertainID on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

Figure 31 Setup CertainID – Finished.

17. During the installation if you chose to create desktop shortcuts, four desktop shortcuts will be created as show in Figure 32.

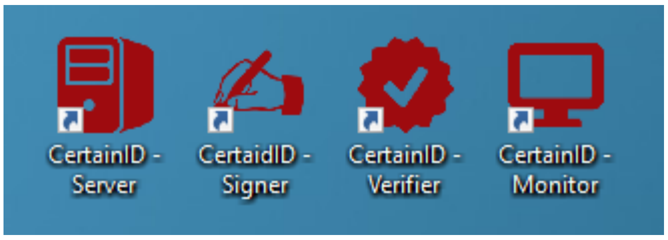
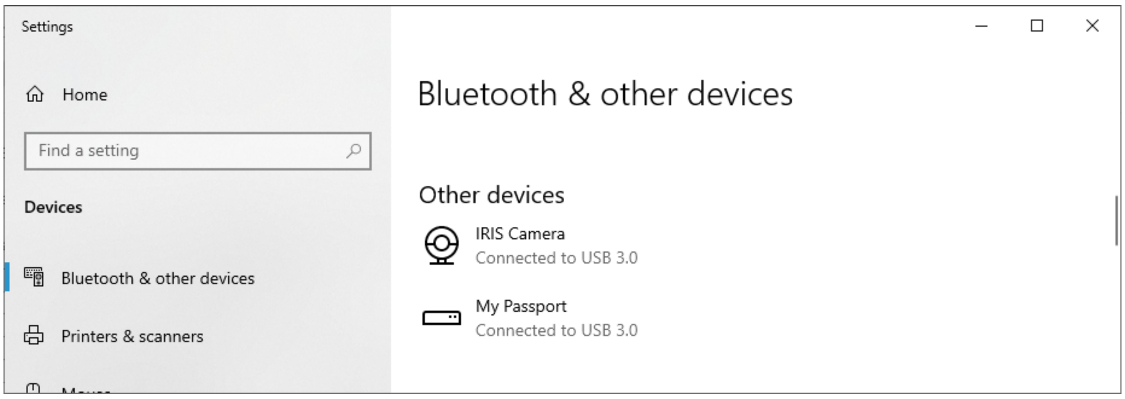


Figure 32 CertainID shortcuts.

3.4 Running


Settings

Home

Find a setting

Devices

Bluetooth & other devices

Printers & scanners

Bluetooth & other devices

Other devices

IRIS Camera
Connected to USB 3.0

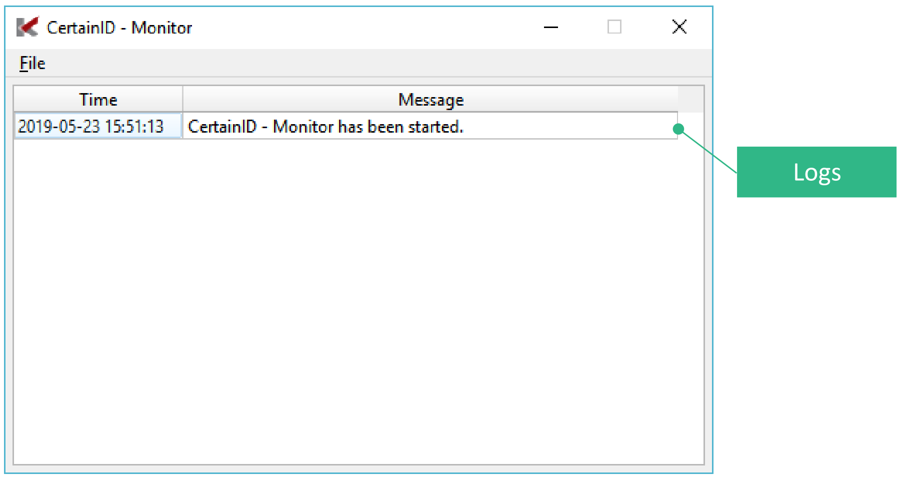
My Passport
Connected to USB 3.0

Figure 33 IRIS Camera installed.

Before running, attach an iris camera which provided by InfinityOptics to your PC. MS Windows will recognise the iris camera automatically. Figure 33 shows the installed iris camera. Once configuration of DB is done, no additional actions are required. Simply copy the CertainID package to an appropriate location and execute the EXE files.

3.4.1 Monitor

To run the Monitor, execute ‘monitor.exe’. The Monitor shows log messages since the Monitor started.

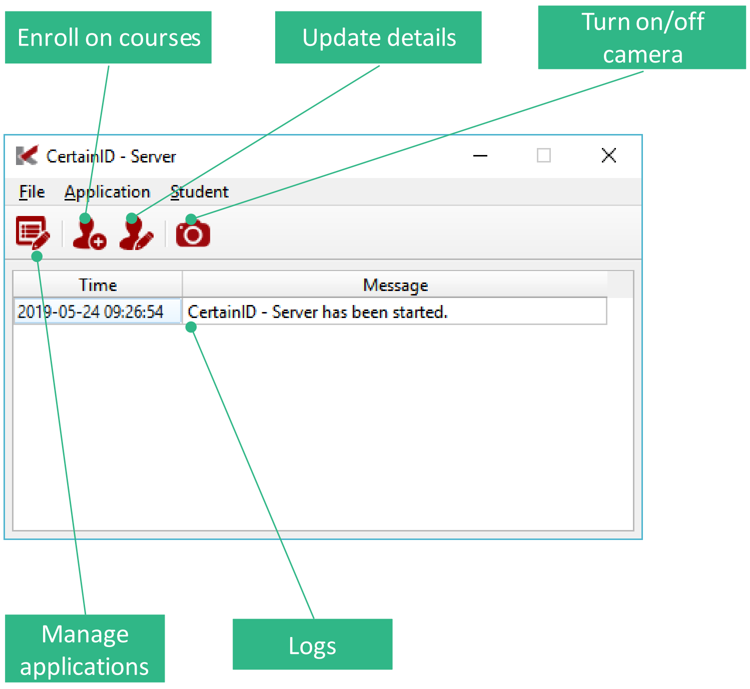


Time	Message
2019-05-23 15:51:13	CertainID - Monitor has been started.

Figure 34 Main window of the Monitor.

3.4.2 Server

To run the Server, execute ‘server.exe’. The Server supports many features; managing applications, enrolling on applications and updating students’ details. Administrators and teachers are main users of this program. The server supports a dummy DLL for a demo without camera module.

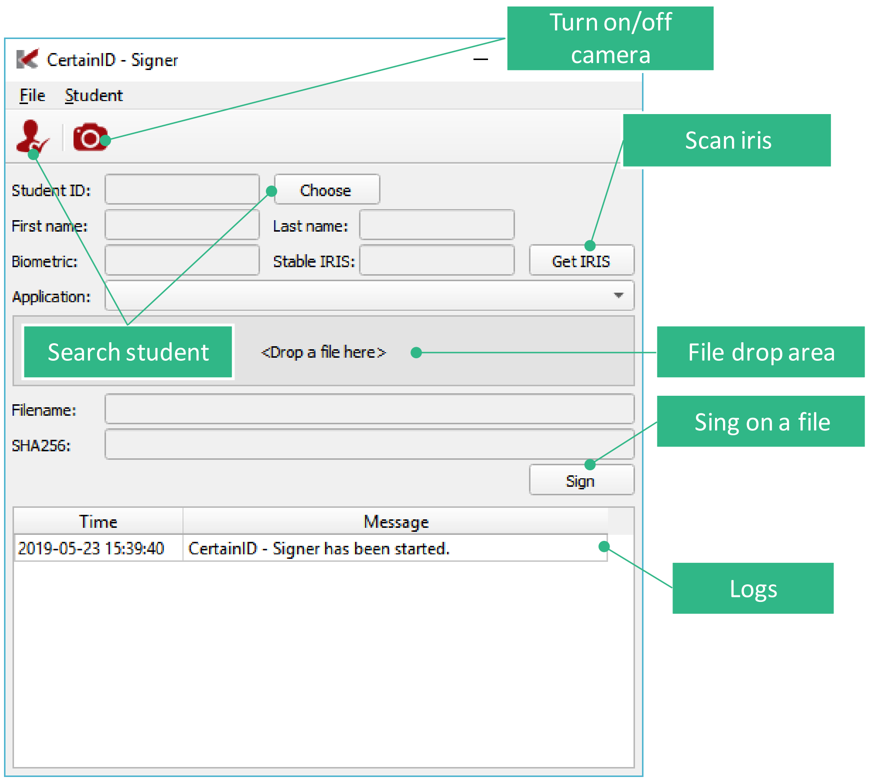


Time	Message
2019-05-24 09:26:54	CertainID - Server has been started.

Figure 35 Main window of the Server.

3.4.3 Signer

To run the Signer, execute ‘signer.exe’. Through the Signer, users can sign on any documents. By dropping a document file on the drop area (please refer Figure 36), a user can sign on the document. The Signer will create a corresponding signature file and a public key file into the same folder which the document file is located. The Signer also supports a dummy DLL for a demo without camera module.

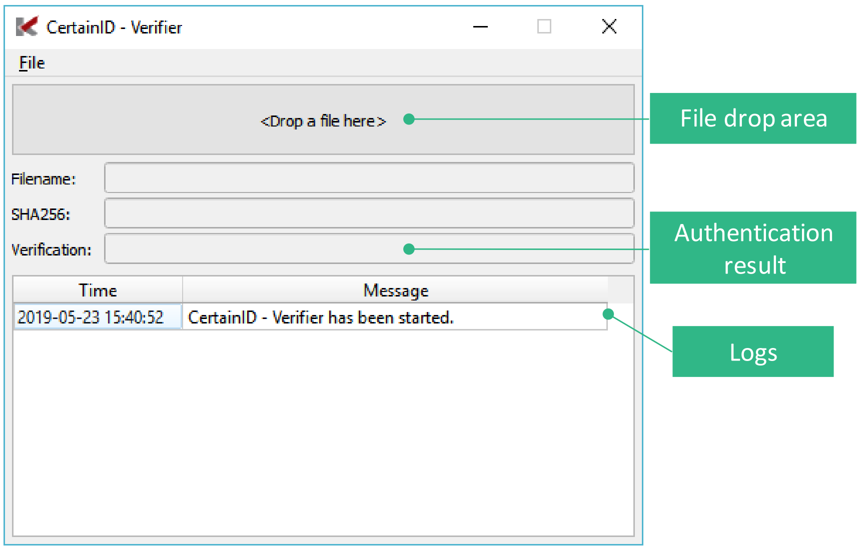


Time	Message
2019-05-23 15:39:40	CertainID - Signer has been started.

Figure 36 Main window of the Signer.

3.4.4 Verifier

To run the Verifier, execute ‘verifier.exe’. The Verifier verifies signed documents. By dropping a signed document file on the drop area (please refer Figure 37), the Verifier will check whether the document is authentic or not. The Verifier automatically will look for signature files and public key files from the same folder which document files are located.



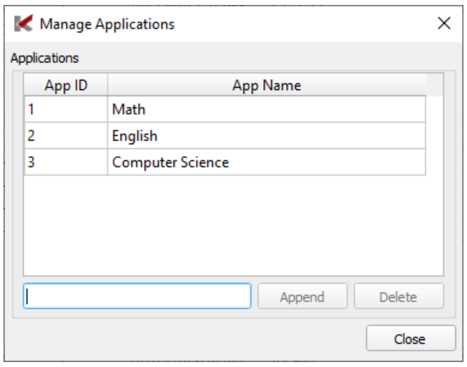
Time	Message
2019-05-23 15:40:52	CertainID - Verifier has been started.

Figure 37 Main window of the Verifier.

3.5 Demonstration

We prepared a demo scenario to show how the CertainID works as the followings:

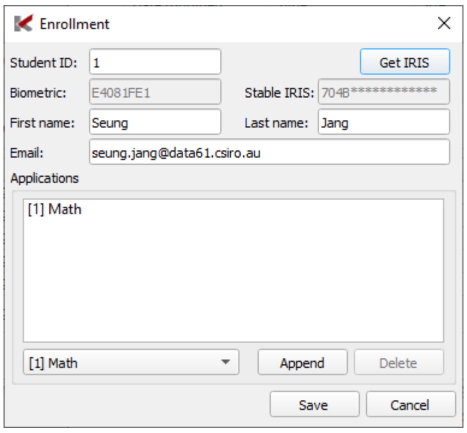
18. Run Monitor
    a. An administrator runs the Monitor to observe the CertainID.
19. Manage applications
    a. An administrator runs the Server.
    b. The administrator reviews application list and modifies the list if necessary.



App ID	App Name
1	Math
2	English
3	Computer Science

Figure 38 Manage Applications Dialog.

20. Enrol on a course
    a. A student comes to the Admin Office for enrolling on a course.
    b. An administrator opens the Server and fills the enrolment form up.


Enrollment

Student ID:	1		Get IRIS
Biometric:	E4081FE1	Stable IRIS:	704B************
First name:	Seung	Last name:	Jang
Email:	seung.jang@data61.csiro.au		

Applications

[1] Math

[1] Math

Figure 39 Enrolment form.

c. The student scans his/her iris. Figure 40 shows an iris camera with a mirror to assist users. For scanning iris properly, locate your face in the centre of the mirror, keep a distance from the mirror about 35cm and look at the camera which located in the middle.

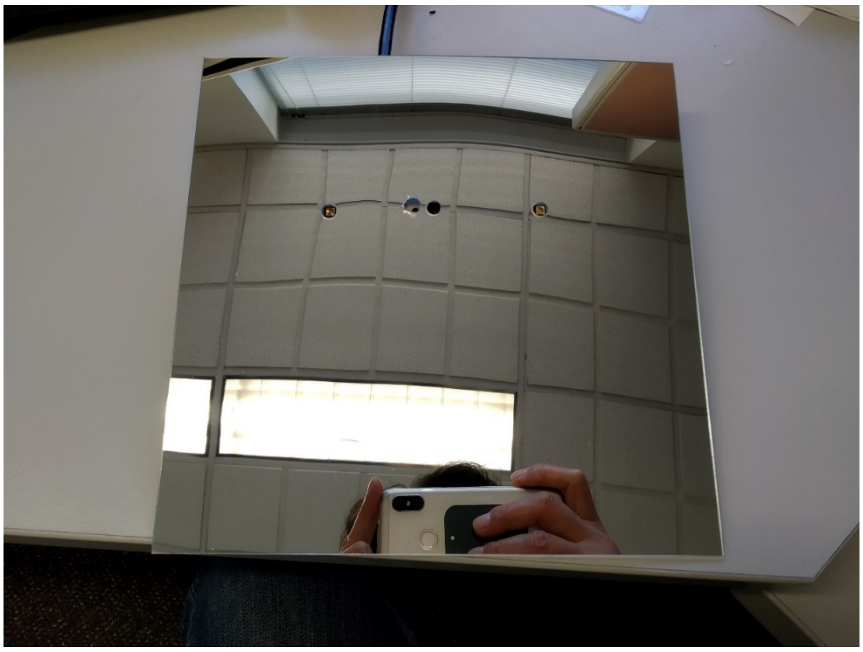


Figure 40 Iris camera with a mirror to assist users.

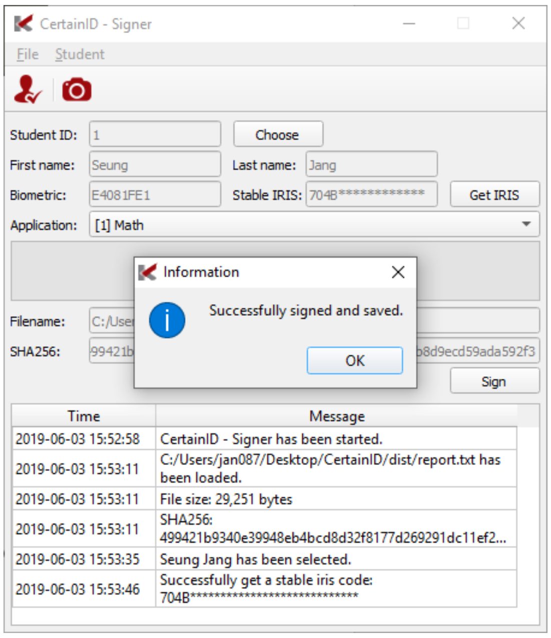
d. The administrator saves the form.

21. Submit a report

a. The student writes a report (any type of electronic assessment such as homework, project, exam, etc.).

b. The student/teacher opens the Signer and drop the report on the drop area in the Signer.

c. The student scans his/her iris.



Student ID: 1
First name: Seung Last name: Jang
Biometric: E4081FE1 Stable IRIS: 704B************
Application: [1] Math
Filename: C:/User
SHA256: 99421b … b8d9ecd59ada592f3

Successfully signed and saved.

Time	Message
2019-06-03 15:52:58	CertainID - Signer has been started.
2019-06-03 15:53:11	C:/Users/jan087/Desktop/CertainID/dist/report.txt has been loaded.
2019-06-03 15:53:11	File size: 29,251 bytes
2019-06-03 15:53:11	SHA256: 499421b9340e39948eb4bcd8d32f8177d269291dc11ef2...
2019-06-03 15:53:35	Seung Jang has been selected.
2019-06-03 15:53:46	Successfully get a stable iris code: 704B****************************

Figure 41 Sign on a document.

d. Submitted report can also be signed by teacher to prevent deny by both parties.

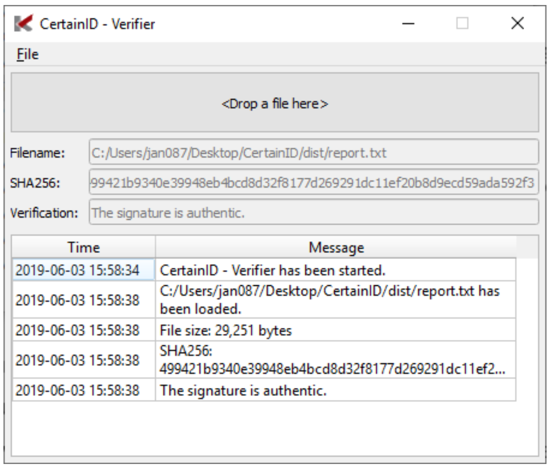
e. The student keeps a copy of report, own signature, own public key, teacher's signature and teacher's public key.

f. The teacher keeps a copy of report, own signature, own public key, student's signature and student's public key.

22. Verify the report

a. Teacher verifies the report before scoring, e.g. if any part is missing or if it has been tempered with in the system.

b. A teacher opens the Verifier and drop a submitted report.



Filename: C:/Users/jan087/Desktop/CertainID/dist/report.txt
SHA256: 99421b9340e39948eb4bcd8d32f8177d269291dc11ef20b8d9ecd59ada592f3
Verification: The signature is authentic.

Time	Message
2019-06-03 15:58:34	CertainID - Verifier has been started.
2019-06-03 15:58:38	C:/Users/jan087/Desktop/CertainID/dist/report.txt has been loaded.
2019-06-03 15:58:38	File size: 29,251 bytes
2019-06-03 15:58:38	SHA256: 499421b9340e39948eb4bcd8d32f8177d269291dc11ef2...
2019-06-03 15:58:38	The signature is authentic.

Figure 42 Verify a signed document

23. Conflict resolution

a. The student claims that the submitted report was modified or lost, or

b. The teacher claims that the student has not submitted a report or part of it.

c. Both parties present and verify the reports and signatures they must resolve the conflict.

4 Conclusion

This project delivers the CertainID solution consisting of a cryptosystem, a software server component for application enrolment, a software signer module for user registration and transaction signing, a software verifier module to check signatures, a software monitor module to demonstrate the interaction between CertainID components, third-party biometric sensor hardware and its APIs.

The CertainID project fills a significant gap by providing a safe method for biometrics-based security solutions. CertainID uses biometric features through a one-way mapping to create a revokable biometric code and uses the biometric code to encrypt a set of randomly generated private/secret keys. Users present their biometric features to unlock a private/secret keys for signing (or decrypting) a transaction.

CertainID ensures that individuals' mass biometric data is not lost/compromised, and no sensitive information associated with individuals can be hacked off the device if the device storing CertainID data is lost or compromised. Accordingly, CertainID helps remove the security key management burden and associated risks both from individuals and organizations.

4.1 Potential Applications

This report illustrates one application of CertainID for educational institutes. CertainID can solve many trust issues in a most useful manner in classrooms relating to scenarios such as identification, authentication, prevention of cheating, and ensuring non-repudiation. The solution applies to all domains and applications involving many users. It solves key management and authentication problems and removes complications resulting from storing many sensitive user credentials. Transactions used by CertainID can be any sequence of bytes such as text or binary files of arbitrary sizes, communication messages, emails, PDF/Word/PS forms and electronic tokens/tickets. From this perspective, CertainID has vast potential for applications in finance, government services, energy, transportation, health, manufacturing and service domain as key and identity management platform. The lightweight characteristic of the solution makes it suitable for mobile to cloud computing systems.

4.2 Future Work

Kollakorn and Data61 have developed the CertainID demonstrator using a low-cost camera capable of reading iris from a comfortable distance (e.g., 40 cm). Future work would extend the existing CertainID demonstrator to a full-fledged cloud service. This would transform CertainID to a biometric ID provider for Kollakorn clients. In this case, the users would use their devices such as a USB/Bluetooth sensors, laptop, tablet or smartphone to read their biometric information, unlock their security keys stored by CertainID service, and use these keys in any security application they wish, e.g., signing a PDF form or a bank transaction, encrypting an email, or signing in a web service.

Abbreviations and Glossary

AES - Advanced Encryption Standard

API - Application Programming Interface, which is a is a set of subroutine definitions, communication protocols, and tools for building software.

BCH - Bose–Chaudhuri–Hocquenghem codes form a class of cyclic error-correcting codes that are constructed using polynomials over a finite field.

CPU - Central Processing Unit

DLL - Dynamic-Link Library, which is a Microsoft implementation of shared library concept.

ECC - Elliptic Curve Cryptosystem

ID - Identity

NIST - The U.S. National Institute of Standards and Technology

RSA - Rivest–Shamir–Adleman which is one of the first public-key cryptosystems and is widely used for secure data transmission

SHA - Secure Hash Algorithm

USB - Universal Serial Bus

FOR FURTHER INFORMATION

Seyit Camtepe
Principal Research Scientist
t +61 2 9372 4325
e seyit.camtepe@data61.csiro.au
w www.data61.csiro.au

Surya Nepal
Senior Principal Research Scientist
t +61 2 9372 4256
e surya.nepal@data61.csiro.au
w www.data61.csiro.au