Analysis of Security Protocols using Finite-State Machines

Dania Aljeaid
School of Science and Technology
Nottingham Trent University
Nottingham, United Kingdom

Xiaoqi Ma
School of Science and Technology
Nottingham Trent University
Nottingham, United Kingdom

Caroline Langensiepen
School of Science and Technology
Nottingham Trent University
Nottingham, United Kingdom

***Abstract*—This paper demonstrates a comprehensive analysis method using formal methods such as finite-state machine. First, we describe the modified version of our new protocol and briefly explain the encrypt-then-authenticate mechanism, which is regarded as more a secure mechanism than the one used in our protocol. Then, we use a finite-state verification to study the behaviour of each machine created for each phase of the protocol and examine their behaviours together. Modelling with finite-state machines shows that the modified protocol can function correctly and behave properly even with invalid input or time delay.**

Keywords—identity-based cryptosystem; cryptographic protocols; finite-state machine

I. Introduction

Security protocols are becoming the core subject in communication systems and verifying them has gained significant attention by researchers and developers. Security analysis aims to formally guarantee these protocols to satisfy their specifications and they can function soundly. Security evaluation is a fundamental step in the development of security protocols. The methods used to analyse security protocols can be categorised into two groups: methods based on analytical approach and methods based on simulation. The analytical approach offers accurate results and provides a clear perception of the system characteristics. However, this approach becomes unreliable when dealing with high complex system. Therefore, the latter approach, simulation approach, has become more popular in system analysis. Simulation tools, such as finite-state machines and Petri nets, expose progress in two directions: one related to the development of faster methods during execution of mathematical algorithms [1], and the other associated with the effectiveness simulation presentations and results [2].

Protocol modelling is a crucial step in designing security protocols. It contributes to diminishing ambiguity and misinterpretation of protocol specifications. For example, modelling a protocol using finite-state machine can help to understand how it will interact with the changes and how it will behave with invalid inputs. A **F**inite-**S**tate **M**achine (FSM) is a powerful tool to simulate software architecture and communication protocols. FSM can only model the control part of a system and consists of a finite number of states, a finite number of events, and a finite number of transitions.

Modelling with finite-state machine helps to understand the behaviour of complex protocol. Also, it offers accurate results and provides a clear perception of the system characteristics. The analysis presented in this paper covers the process of the three-way handshake used to negate the session key and examine the behaviours of the protocol and enumerates all possible states it can reach.

The structure of this paper is organised as follows. In Section 2, we briefly review previous works on extended finite-state machine and briefly discuss the weakness in our new protocol and present modified version of it. In Section 3, we model the modified protocol using EFSM. We then provide a brief discussion on security analysis in Section 4. Finally, the conclusions are given in Section 5.

II. Review of Related Work

A. Extended Finite-State Machines

In order to model the complex behaviour of the proposed protocol, an extended model of FSM is considered. According to [3], EFSM helps to comprehend the *state space* complexity of a system when the number of states and transitions increases Also, they emphasise the importance of introducing *state variables* in FSM models. State variable play a key role in modelling because they can "define a range of arithmetic and logical operators to manipulate state variables and trigger transitions based on logical primitives" [3]. Moreover, EFSM with variables can transfer variable values from one model to another. Consequently, the produced output value from one machine can be consumed by other machines. With the introduction of variables, EFSM allows one to model a system with conditions. Transitions may have guards and predicates, which consist of operations or Boolean-valued expressions that can depend on input variables [3].

A formal definition of an EFSM is as follows [3, 4]:

An Extended Finite State Machine (EFSM) M is a tuple (*S, T, E, V*) where,

S is a set of states,
T is a set of transitions,
E is a set of events, and
V is a store represented by a set of variables.

Transitions have a source state *source(t)* ∈ *S*, a target state *target(t)* ∈ *S* and a label *lbl(t)*. Transition labels are of the form *e1*[*c*]/*a* where e1 ∈ *E*, *c* is a condition and *a* is a sequence of actions.

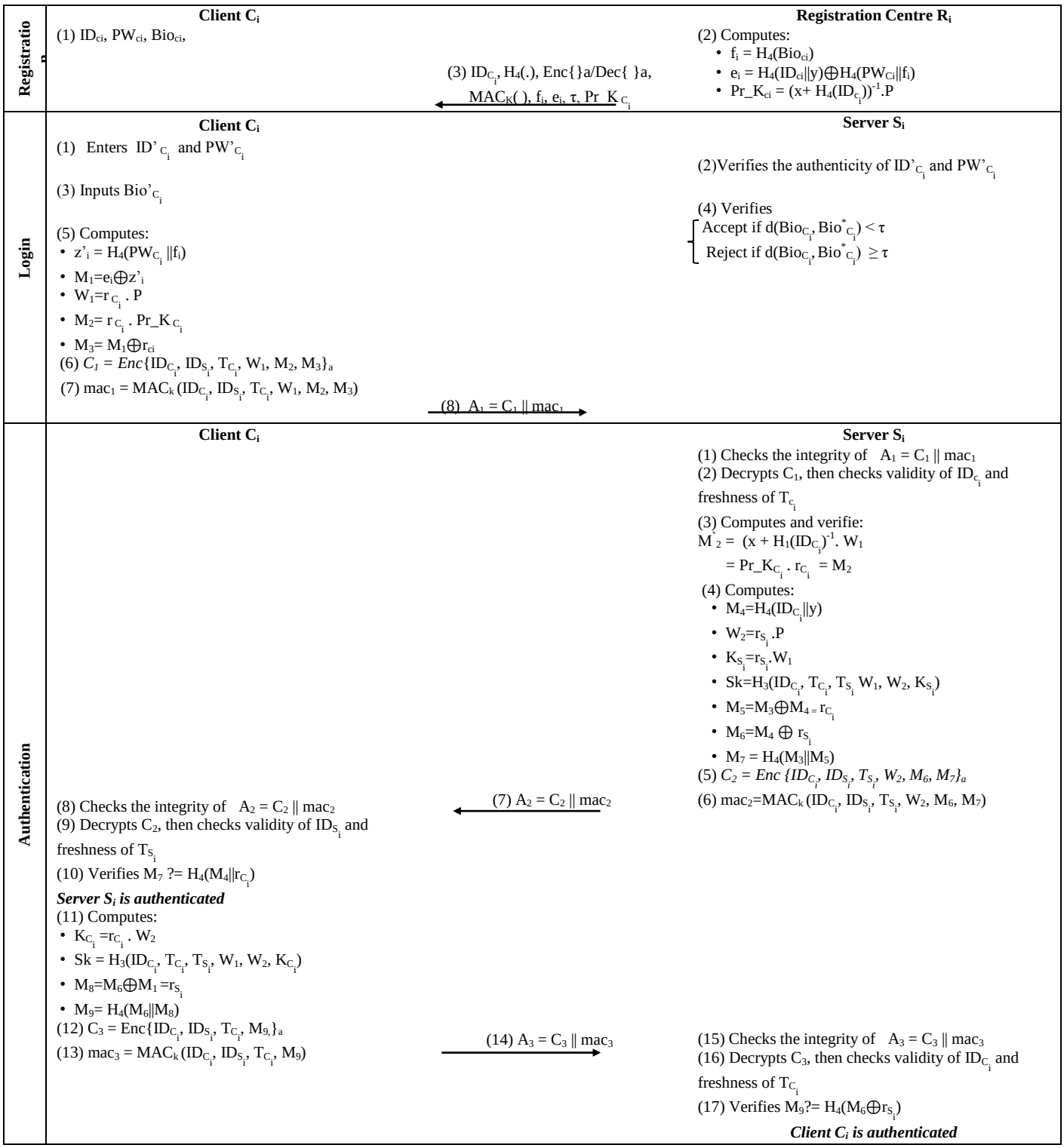

Fig. 1. The modified proposed protocol

B. Review of Proposed Protocol

In our previous work [5], we have developed a new authentication protocol that allows remote mutual authentication with key agreement. Our new protocol is based on biometric verification and ID-based Cryptograph. However, it is not secure against chosen-ciphertext attacks. The new protocol needs modifications to initiate secure authentication between the client and server.

The modified version of the proposed protocol should improve security and provide users with better authentication and data confidentiality. To address and correct the perceived security weakness in the proposed protocol, authenticating the ciphertext by applying encrypt-then-authenticate mechanism is considered to be one of the secure methods for security

protocols. The previous message exchange in the proposed protocol was constructed like this:

Encrypt (Message || MAC)

The new modification for the message exchange will be constructed as this [7,8]:

Encrypt (Message) || MAC

This way the MAC is covering the entire ciphertext to preserve the integrity of the cipher message. The MAC value is then appended to the encrypted message. When the recipient receives the authenticated encrypted message, the MAC should be evaluated before attempting to decrypt the ciphertext. If the MAC verification fails, the recipient will terminate the session immediately. This process will be efficient by eliminating the time spent to going through the manipulated data. The enhancements for the proposed protocol will only affect part of the registration phase and the authentication and key agreement phase. Additionally, enclosing the identity of the server along with the client's identity can mitigate the impact of masquerading attack. The ID's of entities must be unique in the network. Thus, the entities that wish to communicate are aware of each other. The modified protocol is summarised in Fig. 1. Based on the investigation above, we need to modify the state machine described in [5,6] according to the new enhancements.

IV. Protocol Model and State Machine

The EFSM is used to model the communication channel of the proposed protocol between the Client C_i and the Server S_i. Since the exchange of packets follows a pattern defined by a finite set of rules, each principal in the protocol has a corresponding state machine: $EFSM_{server}$, $EFSM_{register}$ and $EFSM_{client}$.

A. Verifier EFSM

The EFSMverifier is an embedded machine within $EFSM_{client}$ and $EFSM_{server}$ where states themselves can have other machines. To be precise, it is a set of sub-states that are integrated as a nested finite state machine which are inside the states S5 and S6 in $EFSM_{server}$ and state C6 in the $EFSM_{client}$ .It is only activated when the authentication and key agreement have started. The $FSM_{verifier}$ is triggered when it obtains authentication information from FSM_{client} or FSM_{server}. It represents various transitions during the authentication and validation process. This machine is modelled using 5 states and 8 transitions. Table 1 describes the transitions specifications and Fig.2 illustrates the verifier modelled by EFSM.

- State V0: this state accepts the authentication information that needs to be verified and sends an authenticity-checking request to V1.
- State V1: the EFSMverifier verifies the integrity of the received cipher message by recalculating the MAC value of the received message and comparing it with the attached MAC value. If the MAC values appeared to be identical, the machine triggers itself to the next state, V2, since the condition is fulfilled. However, if the comparison shows a different result, this would trigger to invalid state that then leads to termination.
- State V2: while in this state, EFSMverifier decrypts the ciphertext since MAC integrity check has been successful. After decryption is successful, the EFSMverifier transitions to the state V3.
- State V3: the $EFSM_{verifier}$ checks the freshness of T via $T` - T_{C_i} \leq \Delta T$. If the freshness is valid, the $EFSM_{verifier}$ triggers itself to the next state. Otherwise, it produces invalid input if the freshness of $T` - T_{C_i} \geq \Delta T$ and changes to state V0.
- State V4: while in state V4, the $EFSM_{verifier}$ checks the validity of ID and based on the result it changes to state V0 either with event of valid ID or invalid ID.

TABLE I. The Transitions Specification of the Verifier EFSM

Transition	Transition Direction	Guards/Condition
Validate	C5 → V0 S5 → V0 S6 → V0	
Authenticity check	V0 → V1	
Invalid	V1 → V0	Client_MAC != Server_MAC
Integrity checked	V1 → V2	Client_MAC== Server_MAC
Decrypted the ciphertext	V2 → V3	
Freshness checked	V3 → V4	$T` - T_{C_i} \leq \Delta T$
Invalid	V3 → V0	$T` - T_{C_i} > \Delta T$
ID valid	V4 → V0	
ID invalid	V4 → V0	Invalid ID

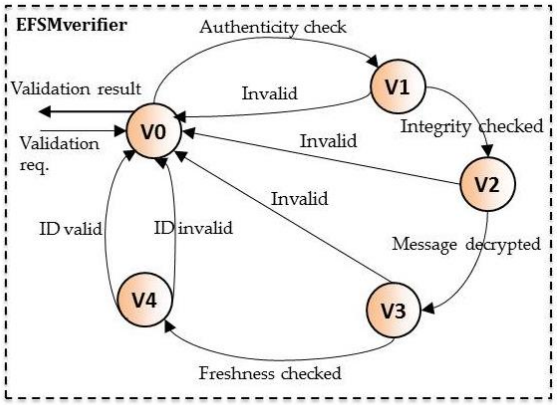

Fig. 2. The verifier machine modelled by EFSM

B. Server EFSM

The FSM at the server side represents the various on-going communications with the client at any point in time. It is modelled using 10 states and 24 transitions and one nested EFSM as detailed below. Table 2 describes the transitions

specifications Fig. 3 shows the transitions diagram for the $EFSM_{server}$.

1) The $EFSM_{server}$ will loop continuously while the server is waiting for clients. The machine advances to the next state once it is triggered by a login/enrol transition.

2) When the $EFSM_{server}$ is in the state S1, it checks the validity of the received ID. If ID is proved to be incorrect, S_i will request C_i to enter the valid ID for three times and $EFSM_{server}$ will loop until C_i enters the valid ID up to three times. In the latter case, the C_i's account will be blocked and $EFSM_{server}$ will change to state S4 from state S1. Generally, three attempts are made through our protocol steps to allow common errors.

3) When the $EFSM_{server}$ is in the state S2, it is triggered by a valid ID and it is now waiting for a valid PW. Once S_i receives PW, it verifies the validity of PW. If PW is proved to be invalid, S_i will request C_i to enter the valid PW for three times and $EFSM_{server}$ will loop until C_i enters the valid PW or if the attempts exceed three times. In the latter case, the C_i's account will be blocked and $EFSM_{server}$ changes state to S4 from state S2.

TABLE II. THE TRANSITIONS SPECIFICATION OF THE SERVER-SIDE EFSM

Transition	Transition Direction	Guards/Condition
Waiting for clients	S0 → S0	
Request to enrol	S0 → R0	ClientEnrol == True
Client is registered	S0 → S1 R0→S0	ClientReg == True
Enter ID	S0 → S1	ID Valid
Enter Password	S1 → S2	Password Valid
Submit Biometric	S2 → S3	Biometric Valid
Request client login (SYN received)	S3 → S5	
Re-enter ID/Password/ Biometric	S2→S2 S3→S3 S4→S4	ID_attempt < 3, ID_attempt = ID_attempt +1 PW_attempt < 3, PW_attempt = PW_attempt +1 Bio_Attempt == < 3, Bio_attempt = Bio_attempt +1
Invalid ID/Password/Biometric	S2→S4 S3→S4 S4→S4 S5→S4 S6→S4	ID_attempt == 3 PW_attempt == 3 Bio_Attempt == 3 Invalid ID
Send SYN/ACK and C2	S5→S6	Validation check is valid
Client ACK and C3 received	S6→S7	Validation check is valid
Terminate	S5→S8 S6→S8	
Timeout	S1→ S0 S2→S0 S3→S0	

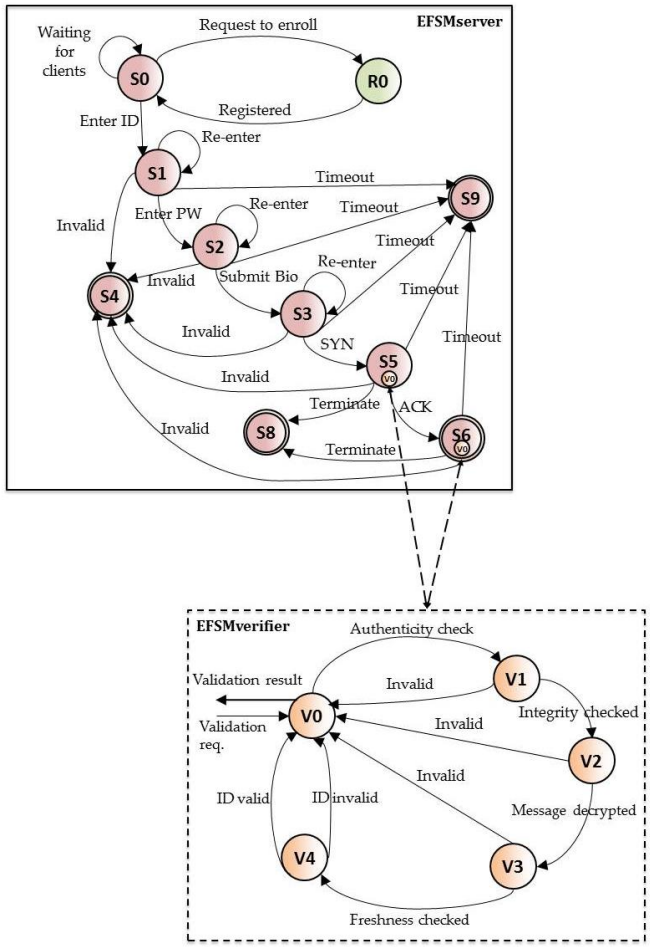

Fig. 3. The server machine modelled by EFSM

4) When the $EFSM_{server}$ is in the state S3, it is triggered by a valid PW and it is now waiting for a valid Bio. Once S_i receives Bio, it verifies the validity of Bio by comparing the imprinted Bio with the template stored. If Bio does not match the stored template, S_i will request C_i to enter the valid Bio up to three times and the $EFSM_{server}$ will loop until C_i enters the valid Bio or if the attempts exceed three times. In the latter case, the C_i's account will be blocked and the $EFSM_{server}$ changes state to S4 from state S3.

5) In state S5, the $EFSM_{server}$ waits until it receives the login request SYN = $A_1 = C_1 \,||\, mac_1$ from the FSM_{client} to establish a connection by performing three-way handshake.

6) While in state S5, the $EFSM_{server}$ activates the nested $EFSM_{verifier}$ and waits for the validation check result.

7) Once the validation has proved to be true. S_i generates a random number and timestamp, then S_i replies with authenticated SYN/ACK = $A_2 = C_2 \,||\, mac_2$ to the $EFSM_{client}$, which is a combination of $C_2 = Enc\ \{ID_{C_i}, ID_{S_i}, T_{S_i}, W_2, M_6, M_7\}_a$ and $Mac_2 = MAC_k(ID_{C_i}, ID_{S_i}, T_{S_i}, W_2, M_6, M_7)$.

8) In state S6, $EFSM_{server}$ waits until it receives ACK from the $EFSM_{client}$. Once the authenticated ACK = $A_3 = C_3 \,||\, mac_3$

is received, the $EFSM_{server}$ activates the nested $EFSM_{verifier}$ and waits for the validation check result.

9) Once the validation check is proved to be true, the $EFSM_{server}$ verifies $M_9 \stackrel{?}{=} H_4\ (M_6 \parallel r_{S_i})$. At this point, S_i authenticates C_i as a legitimate user.

10)At state S5 and state S6, $EFSM_{server}$ terminates the current session if any of the following situations occurs:

- The client ID is invalid
- The freshness of $T` - T_{C_i} \geq \Delta T$
- A negative result when checking the integrity of mac_1 and mac_3
- $M2 != (x + H_1(ID_{C_i})^{-1}. W_1$
- $M9 != H_4 (M_6 \parallel r_{S_i})$

At any stage of $EFSM_{server}$ activity, $EFSM_{server}$ aborts the current session and changes to state S9 if the timeout exceeds the defined TIME_WAIT while waiting for packets. This feature helps to prevent an infinite wait when the $EFSM_{client}$ fails to respond.

C. Client EFSM

The EFSM at the client side represents the various on-going transmissions with the server at any point in time. It is modelled using 9 states, 22 transitions, and one nested EFSM as detailed below. Fig. 4 shows the transition diagram for the $EFSM_{client}$ and Table 3 describes the transitions specifications.

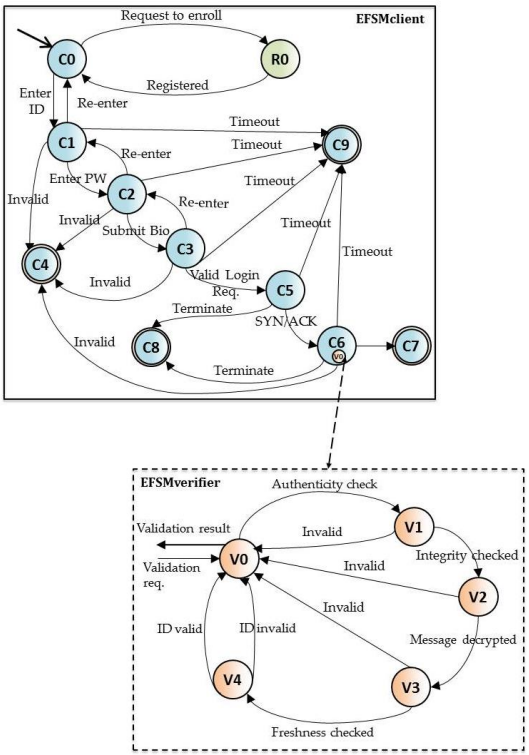

Fig. 4. The client machine is modelled by EFSM

TABLE III. THE TRANSITIONS SPECIFICATION OF THE CLIENT-SIDE EFSM

Transition	Transition Direction	Guards/Condition
Request to enrol	C0 → R0	ClientEnrol == True
Client is registered / Enter ID	C0 → C1	ClientReg == True
Enter Password	C1 → C2	ID valid
Submit Biometric	C2 → C3	Password valid
Send login request SYN (C_1)	C3 → C5	Biometric valid
Re-enter ID/Password/ Biometric	C1→C1 C2→C2 C3→C3	ID_attempt < 6, ID_attempt = ID_attempt +1 PW_attempt < 3, PW_attempt = PW_attempt +1 Bio_Attempt < 3, Bio_attempt = Bio_attempt +1
Invalid ID/Password/Biometric	C1→C4 C2→C4 C3→C4	ID_attempt == 6 PW_attempt == 3 Bio_Attempt == 3
Client receives SYN/ACK (C2)	C5→C6	
Client sends ACK (C3)	C6→C7	Validation check is valid
Authenticated by server	C7→C8	
Terminate	C5→C8 C6→C8	
Timeout	C1→C0 C2→C0 C3→C0	

1) First, the $EFSM_{client}$ is in the initial state C0. That is when the request for register/login is initiated by itself. While in state C0, the $EFSM_{server}$ checks whether C_i is enrolled or not. The next state will be determined according to the condition ClientReg == True.

2) In state C1, C2, C3, the FSM_{client} is waiting for validating ID, PW, and Bio. Once the client credentials are validated, the $EFSM_{client}$ triggers itself and changes to state C5.

3) In states C1, C2, C3, the client may be required to re-enter ID, PW, Bio in cases where they were incorrect. However, the client's account will be blocked if the number of attempts exceeds three, which changes the above states to state C4.

- ID_attempt < 3, ID_attempt = ID_attempt +1
- PW_attempt < 3, PW_attempt = PW_attempt +1
- Bio_Attempt < 3, Bio_attempt = Bio_attempt +1

4) In state C5, The $EFSM_{client}$ generates a random number and a timestamp to calculate the encrypted login request $\{ID_{C_i}, ID_{S_i}, T_{C_i}, W_1, M_2, M_3\}_a$ and then computes $mac_1 = MAC_k (ID_{C_i}, ID_{S_i}, T_{C_i}, W_1, M_2, M_3)$. It sends $A_1 = C_1 \parallel mac_1$ to the $EFSM_{server}$. This request represents the SYN part in the three-way handshake procedure.

5) While in state C5, the FSM_{client} is waiting for the $EFSM_{server}$ to respond after sending the login request to establish the connection. Once the authenticated SYN/ACK = $A_2 = C_2 \,||\, mac_2$ is received, the FSM_{client} changes to state C6.

6) In state C6, The $EFSM_{client}$ activates the nested $EFSM_{verifier}$ and waits for the validation check result. Once the validation check is proved to be true, the $EFSM_{client}$ is validating the $EFSM_{server}$ response $M_7 \stackrel{?}{=} H_4 (M_4 \,||\, r_{C_i})$. If S_i is proved to be honest, C_i authenticates S_i at this stage.

7) While in state C6, the $EFSM_{client}$ computes the shared session key $sk = H_3(ID_{C_i}, T_{C_i}, T_{S_i}, W_1, W_2, K_{C_i})$ and finalises the handshake procedure by sending authenticated encrypted $ACK = A_3 = C_3 \,||\, mac_3$ to S_i, which is a combination of $C_3 = Enc\{ID_{C_i}, ID_{S_i}, T_{C_i}, M_9\}_a$ and $Mac_3 = MAC_k (ID_{C_i}, ID_{S_i}, T_{C_i}, M_9)$.

8) In state C7, the $EFSM_{client}$ is waiting to be authenticated by S_i.

9) In state C8, the client terminates the current session if one of the following occurs:

- Negative result when checking the integrity of mac_2
- $T` - T_{S_i} \geq \Delta T$
- The server ID is invalid
- $M_7 \neq H_4 (M_4 \,||\, r_{C_i})$

D. Register EFSM

The EFSM at the registration side represents the various on-going transmissions with the server and client at any point in time. It is modelled using EFSM with 4 states and 7 transitions. Fig. 5 shows the states and transitions diagram for the $EFSM_{register}$.

1) First, the $EFSM_{register}$ is triggered if the client is not enrolled at state R0. That is when the request for registration is initiated by $EFSM_{client}$. While in state C0, the $EFSM_{server}$ checks whether C_i is enrolled or not.

2) Once C_i enters ID, $EFSM_{register}$ changes to state R1 and validates the format of ID. Then $EFSM_{register}$ triggers itself asking C_i to enter PW and changes to state R2.

3) In state R2, on receiving PW for the first time, $EFSM_{register}$ requires C_i to re-enter PW for confirmation. Then it triggers itself and changes to the state R3.

4) In state R3, C_i is required to submit multiple scans of the biometric data to increase accuracy. Once the acquisition process is complete, $EFSM_{register}$ triggers itself and sends a message to $EFSM_{register}$, which indicates that the enrolment is successful.

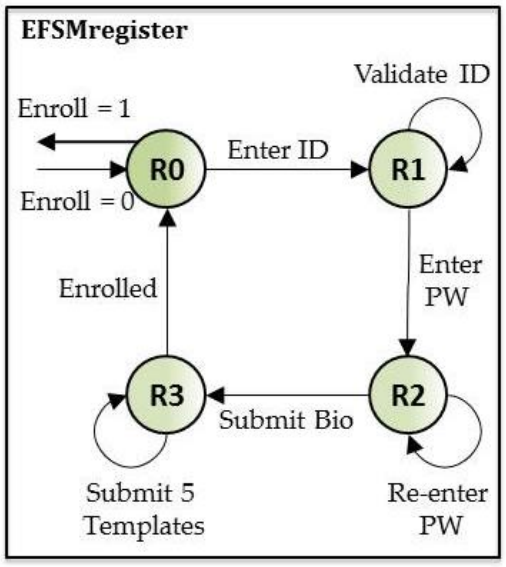

Fig. 5. The client machine is modelled by EFSM

III. Security Analysis

The capability to detect errors and vulnerability is substantial in protocol design implementation. Since communication protocols are partially specified, the finite state approach provides a flexible way to handle invalid inputs and ambiguous specifications, which are usually unspecified or vague in protocol design. Testing the proposed protocol with FSM helps to verify whether the protocol complies with its specification or not. Modelling with FSM shows that the proposed protocol can function correctly and behave properly even with invalid input or time delay.

The state machine in Fig. 6 represents the result of combing the three machines together. The composite model executes efficiently and handles errors in a safe way and it performs certain actions in case of unreliable state. Each valid and reachable state generates a valid protocol state and the transitions can be triggered by either events or guards. Based on the equivalent behaviour, each machine may follow nondeterministic behaviour and produce different outputs according to the original input. For example, if $EFSM_{client}$ generates an illogical input for the authentication process then $EFSM_{client}$ rejects the session and goes to the *terminate state*. Predicating and considering all possible combinations of both desirable and undesirable states are one mean to fully understand the complexity of the proposed protocol.

Note that the states S9 and C9 are defined in terms of a timeout being reached with an inability to complete the mutual authentication.

The states S4 and C4 are defined in terms of an invalid input being injected due to invalid ID, wrong password, or unmatched biometric. The states S8 and C8 are defined in the case of unreliable actions being performed for example, if the integrity or validity check failed. Furthermore, a state machine hierarchy or hierarchical FSM is used to provide a more concrete level of refinement; $FSM_{register}$ can be refined by introducing an "Enrol" feature. This state determines if the client is pre-enrolled or not. The state R0 becomes a new EFSM with three states R1, R2, R3 as described previously.

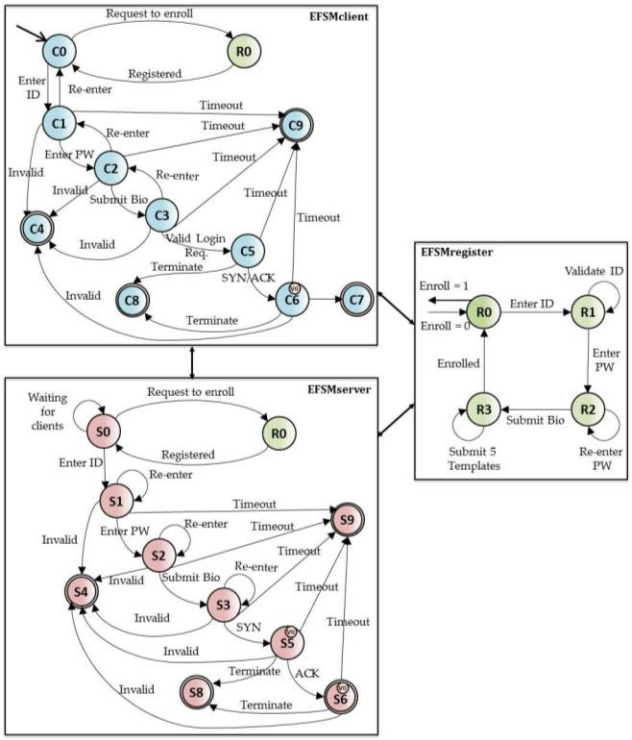

Fig. 6. The modified protocol modelled by EFSM

Based on the parallel behaviour, each machine goes through stages until it reaches the final state. For example, after successful authorisation, the $EFSM_{client}$ switches to the authorised state and proceeds to reach the next state, which is authentication. This comprehensive analysis distinguishes three types of errors that can be detected the protocol run:

- Type I: Timeout errors

This error occurs when the waiting time exceeds the predefined time interval or it occurs when the freshness check exceeds ΔT.

- Type II: Invalid errors

This error is generated in case of invalid inputs, for example, invalid ID, invalid password, or invalid biometric.

- Type III: Terminate error

This error detects if something suspicious occurs in cases where the values did not match. A typical example of this error can be found in the integrity check, when the recomputed MAC value does not match the received MAC value. Another example is when there is a discrepancy in the results of the following equations:

- $M_2 \neq (x + H_1(ID_{C_i})^{-1}. W_1$
- $M_7 \neq H_4(M_3 \| r_{C_i})$
- $M_9 \neq H_4(M_6 \| r_{S_i})$

This error can pose serious threat because it would occur if the data has been modified or injected.

IV. Conclusions

This paper started by giving a brief definition of extended finite state machines (EFSM). Then it elaborates the details of the finite-state verification of the modified protocol and identifies the functionality of each phase. Also, it studies the behaviour of each machine created for each phase and how they interrelate.

The composite model executes efficiently and handles error in a safe way according to their types. The modified protocol connection progresses from one state to another based on the data pertained from the message exchanged. EFSM helps to understand the behaviour of the protocol and logs any unwanted behaviours. This mechanism is very useful for determining the types of errors the protocol experiences during running and it can be useful to later on investigate what causes these errors and learn from them.

In future, an in-depth security analysis and evaluation will be conducted via **P**etri **N**et (PN). PN will be used to simulate the communication patterns between the server and the client as well as to validate the protocol functionality. First, we will model the protocol without an intruder. Then, we will add the intruder to the model and implement a token-passing scheme. At this stage, we will test different attacks, such as impersonation attack, man-in-the-middle attack, and replay attack against the modified protocol and verify the security requirements. After analysis with PN, we will do a comparison between the previous protocol [5] and the modified version of it.

Acknowledgment

This research has been funded by Saudi Arabian Cultural Bureau in London and King Abdul Aziz University in Saudi Arabia.

References

[1] Chiola, G. and Ferscha, A., 1993. Distributed simulation of Petri nets. *IEEE Concurrency*, **1**(3), pp. 33-50.

[2] Genter, G., Bogdan, S., Kovacic, Z. and Grubisic, I., 2007. Software tool for modeling, simulation and real-time implementation of Petri net-based supervisors, *Control Applications, 2007. CCA 2007. IEEE International Conference on* 2007, IEEE, pp. 664-669.

[3] Androutsopoulos, K., Clark, D., Harman, M., Li, Z. and Tratt, L., 2009. Control dependence for extended finite state machines. Fundamental Approaches to Software Engineering. Springer, pp. 216-230.

[4] Alagar, V.S., 2011. Specification of software systems. 2nd edn. England: Springer.

[5] Aljeaid, D., Ma, X. and Langensiepen, C., 2014. Biometric identity-based cryptography for e-Government environment, *Science and Information Conference (SAI), 2014* 2014, IEEE, pp. 581-588.

[6] Aljeaid, D., Ma, X. and Langensiepen, C., Modelling and Simulation of a Biometric Identity-Based Cryptography. *International Journal of Advanced Research in Artificial Intelligence (IJARAI)*, **3**(10),.

[7] KRAWCZYK, H., 2001. The order of encryption and authentication for protecting communications (or: How secure is SSL?), *Advances in Cryptology—CRYPTO 2001* 2001, Springer, pp. 310-331.

[8] KATZ, J. and LINDELL, Y., Introduction to Modern Cryptography 2007.