

Remote User Authentication using Camellia Encryption for Network based applications

Nithishma A¹, Aayush Vijayan², Diya Nanda³

Under the guidance of

Dr. Ch. Aswani Kumar

Abstract

Network based applications are essential for the proper functioning of any institution in this modern era. Almost every single organization, big or small, essential or nonessential utilizes network based applications. With all these important communications and transactions being done on a network, the security and the integrity of the data shared becomes of paramount importance. Larger networks mean a greater number of possible vulnerabilities that could be exploited by anyone with malicious intent. There is an urge to reduce such security breaches and the reasons that allow it. The problem can be resolved by implementing a secure authentication process for accessing the network. This paper proposes remote user authentication using Camellia encryption for Network based applications. The paradigm is efficient and resistant to security attacks. Attacks by third-party are identified by the model and result in the termination of the breach. Thus, only authenticated and verified users would have the access to the system and data. This paper proposes Camellia encryption algorithm as it has specifications similar to AES's. It has remarkable efficiency on hardware as well as software platforms. It is a highly secure cipher with a feistel structure. Furthermore, Camellia also proves to be an efficient cipher to run on multiple platforms.

Keywords: Remote user authentication, Camellia, Network based applications, AES, attacks

1. Introduction

Encryption is an absolute necessity in modern world communications. Huge networks have a large number of devices connected to it, which increases the network's vulnerability to attacks. This puts the user data and communications at the risk of being intercepted. Encryption is the key to ensuring safe communication and transactions. Basically, encryption takes a meaningful piece of text and turns it into something that would be useless to the unauthorized individual. There are various encryption algorithms available to us, but this paper focuses mainly on Camellia encryption algorithms and its applications.

Camellia cipher is symmetric key block cipher. The block size is 128 bits and the key sizes are variable and can be 128, 192 and 256 bits long. Camellia has either 18 rounds (for 128-bit keys) or 24 rounds (for 192- or 256-bit keys) Feistel cipher. After every six rounds of F function, The inverse of the FL-function or a logical transformation layer is applied to the the FL-function. Camellia uses 4 8×8-bit S- boxes. Camellia also uses input, output key whitening. The diffusion layer in Camellia uses a linear transformation which is completely based on a matrix with a determinant quantity of 5. Thus, Camellia Encryption Algorithm is said to be infeasible to any brute force attacks.

User authentication is the premise for most styles of access manipulation and for consumer accountability. There are stages of user authentication.

- Identification step
- Verification step

User authentication is a critical step that has to be considered and evolved for secure communication. So, authentication is required to achieve confidentiality and make the network stable from attacks. One such mutual authentication is advanced in this paper which permits secure information transmission. We've discussed the use of Camellia for a remote user authentication scheme that provides mutual authentication using symmetric keys. This protocol is later discussed in detail.

This paper focuses on the possibilities of applications of Camellia in various authentication schemes. We discuss the related work that has been done in the field. This paper then discusses the Camellia algorithm in detail and the qualities that make it a suitable alternative to its more popular contemporaries like AES or DES. We also propose a remote user authentication scheme implemented using Camellia. We then provide a comparative analysis of a few popularly used encryption algorithms and how they rank against Camellia.

2. Related Work

The paper by Kazumaro Aoki et al. in [1] suggests that the encryption algorithm Camellia has efficiency on software as well as hardware platforms. Camellia cipher has a very conservative design, it is highly secure. Various attacks like brute force, boomerang, Interpolation, Square, linear sum and differential attacks have been tested on Camellia cipher and it has been claimed that Camellia has a very strong security against differential and linear cryptanalysis. As AES algorithm, Camellia offers encryption speeds that are similar to those of AES in multiple software and hardware platforms. The paper by M. Matsui et al. in [2] implements Camellia encryption algorithm as it has the same interface and almost the standards as of AES algorithm. Camellia is proven to have excellent key setup time and Camellia's key agility is far better than that of AES. The security of Camellia has been evaluated by using the best cryptanalytic techniques. It has been confirmed that the cipher has no linear and differential attacks which have a probability that is greater than 2^{-128} . This states that it's extremely not likely that these attacks might succeed against the algorithm. Camellia's design offers security against many other advanced cryptanalytic attacks, for example, higher order differential attacks, truncated differential attacks, related-key attacks, interpolation attacks. The paper by Nihong Wang in [3] claims that states that the probability of information repeated in plain text results in a weak cipher, which might make an entry point of security systems for the illegal information attackers. This paper proposes a time-variable algorithm for information encryption based on via work on issues related to persistent plain-text encryption, Camellia algorithm can not only effectively address issues such as unauthorized information theft. Camellia can effectively avoid other password attack methods such as differential failure analysis, thereby improving the security of the entire system and creating a safer information transmission environment. Thus the algorithm is built in such a way that it has a time-varying variable. The paper by Masayuki & Matsumoto et.al. in[4] shows (from the developer's point of view) that Camellia ciphers with a round count greater than 11 are safe against any kind of truncated differential cryptanalysis even if weak-key FL or FL⁻¹-functions are utilized using Matsui's search algorithm. This does not prove that the 10-round Camellia variant is susceptible to truncated differential cryptanalysis. Camellia's auxiliary functions and the S-box make the algorithm unique, strong from differential cryptanalysis and hard to break. The paper by Keerti Srivastava in[5] illustrates the need for remote user authentication. The scheme is based on a public key cryptosystem developed by ElGamal (1985). The scheme does not require a system to keep a table of passwords to verify the users' legitimacy. Furthermore, our scheme can withstand attack replaying messages. Tests have shown that since improving user security for the model using smart cards the attacks and data breaches have that considerably. The scheme can withstand most of the attacks from crypts.

This paper explains the implementation of Camellia for remote user authentication using denning-sacco protocol. Camellia has been claimed as one of the best feistel ciphers as it is highly secure from cryptanalysis. Camellia is also suitable for multiple software and hardware platforms. The computational speed of Camellia is similar to that of AES and excellent key setup time. Also, Camellia's key agility is very superior to that of AES. Although a weak key is provided, Camellia has the ability to have a secure transmission due to the design of the auxiliary function.

3. Camellia for Remote Authentication

Remote Authentication is critical for most network based systems. For our application, we've proposed using Camellia encryption algorithm for the encryption/decryption steps involved in the mutual authentication process. The protocol involved is explained in detail later. In the authentication process, all the data transferred between users and the server is to be encrypted using Camellia and a symmetric key.

3.1 Camellia Encryption

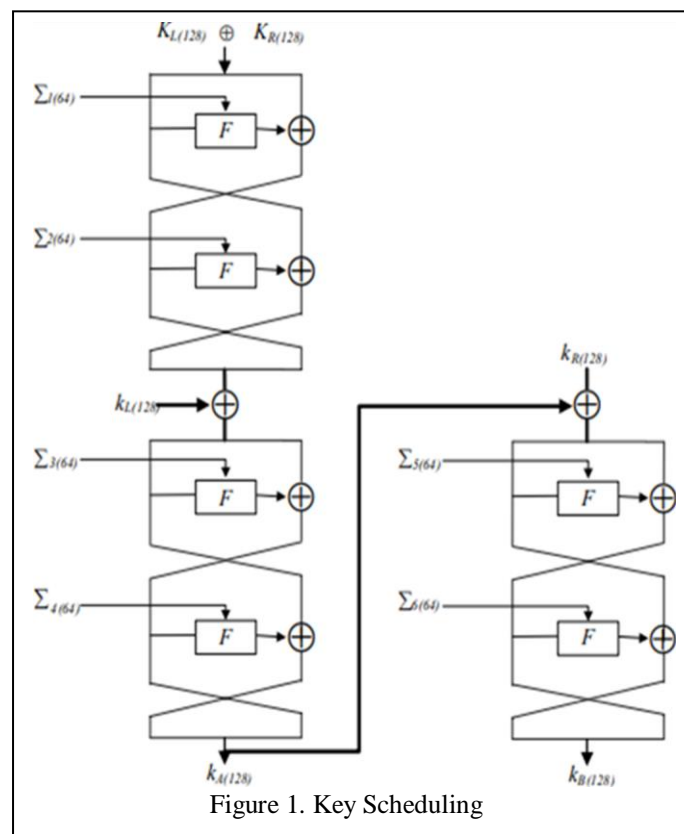
As with any encryption methodology, there are two inputs required: the plaintext and the secure key. In this case, the text needs to be 128 bits long and the key is 128/196/256 bits.

Key scheduling

The key is split into two halves namely the left key(K_L) and the right key(K_R).

- 128 bit key, K_L has the 128bits and K_R has zero bits.
- 192 bit key, K_L has the leftmost 128-bits of the key and the K_R could have the concatenation of rightmost 64-bits of key and the complement of that 64-bits
- 256 bit key, the leftmost 128-bits of key might be present in K_L at the same time as the rightmost 128-bits can be found in K_R .

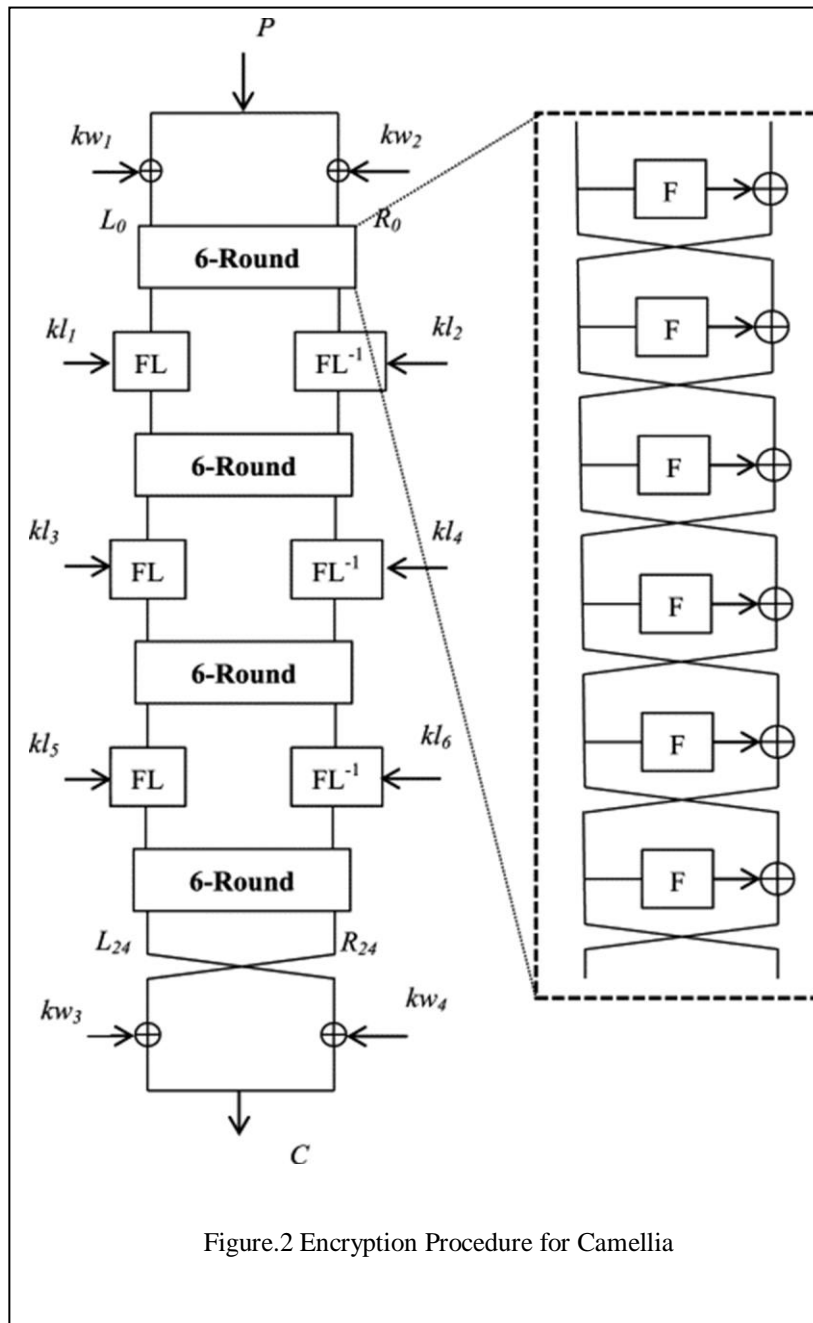
Two extra keys of size 128-bits are generated from the (K_R, K_L) namely K_B and K_A . Two more 64-bit sub keys will be generated by way of rotating the K_R , K_B , K_L , and K_A and extracting the right or left-half of them. These sub keys are used for every round function.



3.1.1 Data randomizing

A 128-bit plain text is split into two blocks of sixty four bits i.e left half of and right half. Encryption is performed on this plaintext of block size 128 using an 18- round Feistel structure.

After every six rounds, a round function(FL) and inverse spherical functions(FL-INV) are inserted.



where,

kl_i: 64-bit subkeys ,

kw_i: whitening keys ,

FL: round function,

FL⁻¹: inverse round function.

L : Left sub half

R : Right sub half

P : Plain text

C : Cipher text

As shown in fig.2, the whole process is as follows.

Let T1 be left half and T2 be right half , F is the round function , k_i are keys for each round

```
T1 = T1 ^ kw1;
T2 = T2 ^ kw2;
T2 = T2 ^ F(T1,k1);
T1 = T1 ^ F(T2,k2);
T2 = T2 ^ F(T1,k3);
T1 = T1 ^ F(T2,k4);
T2 = T2 ^ F(T1,k5);
T1 = T1^F(T2, k6); //round 6
```

```
T1 = FL (T1, kl1);
T2 = FL-1(T2, kl2);
T2 = T2 ^ F(T1, k7);
T1 = T1 ^ F(T2, k8);
T2 = T2 ^ F(T1, k9);
T1 = T1 ^ F(T2, k10);
T2 = T2 ^ F(T1, k11);
T1 = T1 ^ F(T2, k12); //round 12
```

```
T1 = FL (T1, kl3);
T2 = FL-1(T2, kl4);
T2 = T2 ^ F(T1, k13);
T1 = T1 ^ F(T2, k14);
T2 = T2 ^ F(T1, k15);
T1 = T1 ^ F(T2, k16);
T2 = T2 ^ F(T1, k17);
T1 = T1 ^ F(T2, k18);
T2 = T2 ^ kw3; // Post Whitening
T1 = T1 ^ kw4;
```

128-bit ciphertext C is therefore deducible generated from T1 and T2 as follows.

$$C = (T2 \ll 64) \mid T1;$$

The round function uses 8 S-boxes with 8 bit input and 8-bit output. The subkey length is 64bit. The result from the round function is 64 bit which is XORed with the left half and swapped to perform the next round . Thus, the resulting ciphertext has undergone many transformations and is not easy to break. Unlike the 128 bit key, there is an additional set of 6 rounds for 192/256 bit keys.

3.1.2 S-function

S-function is called from the F-function. The size of the input 64 bits is substituted by another 64 bit string, which is returned for further processing. The function utilizes 4 substitution tables also known as s-boxes. The input is divided into 8 bytes x_1, \dots, x_8 . Each of the present s-blocks changes, eight bits received into eight other bits as mentioned by the table. The s-blocks that are written as s_1, \dots, s_4 . If y_1, \dots, y_8 are the eight respective output bytes

$$\begin{aligned} s_2(x) &:= (s_1(x) \lll 1) \\ s_3(x) &:= (s_1(x) \ggg 1) \\ s_4(x) &:= (s_1(x) \lll 1) \end{aligned}$$

112	130	44	236	179	39	192	229	228	133	87	53	234	12	174	65
35	239	107	147	69	25	165	33	237	14	79	78	29	101	146	189
134	184	175	143	124	235	31	206	62	48	220	95	94	197	11	26
166	225	57	202	213	71	93	61	217	1	90	214	81	86	108	77
139	13	154	102	251	204	176	45	116	18	43	32	240	177	132	153
223	76	203	194	52	126	118	5	109	183	169	49	209	23	4	215
20	88	58	97	222	27	17	28	50	15	156	22	83	24	242	34
254	68	207	178	195	181	122	145	36	8	232	168	96	252	105	80
170	208	160	125	161	137	98	151	84	91	30	149	224	255	100	210
16	196	0	72	163	247	117	219	138	3	230	218	9	63	221	148
135	92	131	2	205	74	144	51	115	103	246	243	157	127	191	226
82	155	216	38	200	55	198	59	129	150	111	75	19	190	99	46
233	121	167	140	159	110	188	142	41	245	249	182	47	253	180	89
120	152	6	106	231	70	113	186	212	37	171	66	136	162	141	250
114	7	185	85	248	238	172	10	54	73	42	104	60	56	241	164
64	40	211	123	187	201	67	193	21	227	173	244	119	199	128	158

Figure.3 The S-box S_1

3.1.3 P-function

P-function is also called from inside the F-function, P-function input is of 8 bytes (output of the S-function), modifies them and returns an 8-byte long output. Outputs of P and F function are the same. The input for the P-Function is divided into eight bytes x_1, \dots, x_8 . If y_1, \dots, y_8 are the eight corresponding output bytes, operations performed by the P-function can be described as:

$$y_1 = x_1 \text{ XOR } x_3 \text{ XOR } x_4 \text{ XOR } x_6 \text{ XOR } x_7 \text{ XOR } x_8$$

$$y_2 = x_1 \text{ XOR } x_2 \text{ XOR } x_4 \text{ XOR } x_5 \text{ XOR } x_7 \text{ XOR } x_8$$

$$y_3 = x_1 \text{ XOR } x_2 \text{ XOR } x_3 \text{ XOR } x_5 \text{ XOR } x_6 \text{ XOR } x_8$$

$$y_4 = x_2 \text{ XOR } x_3 \text{ XOR } x_4 \text{ XOR } x_5 \text{ XOR } x_6 \text{ XOR } x_7$$

$$y_5 = x_1 \text{ XOR } x_2 \text{ XOR } x_6 \text{ XOR } x_7 \text{ XOR } x_8$$

$$y_6 = x_2 \text{ XOR } x_3 \text{ XOR } x_5 \text{ XOR } x_7 \text{ XOR } x_8$$

$$y_7 = x_3 \text{ XOR } x_4 \text{ XOR } x_5 \text{ XOR } x_6 \text{ XOR } x_8$$

$$y_8 = x_1 \text{ XOR } x_4 \text{ XOR } x_5 \text{ XOR } x_6 \text{ XOR } x_7$$

3.1.4 F-function

F-function is one of the main capabilities. It is used at some stage in encryption and decryption processes, and for the duration of growing subkeys. The input variable Y of size of 64 bits is combined with one of the present subkeys k of the exact same length. F function returns a sixty four-bit long output block Z .

Data bits are XOR'd to key bits and the result is changed by using two features S and P .

$$(Y, k) \rightarrow Z \Rightarrow P(S(Z \text{ XOR } k))$$

3.1.5 FL-function

FL-function is called during both encryption , decryption processes. The input to the function is of 64 bits. The input data along with one of the subkeys is taken and then certain modifications are performed on it, finally returning a block of data also 64 bits in length. Let the input data block be A, while B being the 64-bit long output block. k_1 is one of the subkeys previously developed:

$$(A, k_1) \rightarrow B \Rightarrow (A_L \parallel A_R, k_{1L} \parallel k_{1R}) \rightarrow B_L \parallel B_R$$

Initially A is divided into two equal parts of 32 bits in length: A_L comprises the left bits of A, and A_R the right bits of A. Once this is over, two new blocks of 32-bits are calculated:

$$B_R = ((A_L \text{ AND } k_{1L}) \lll 1) \text{ XOR } A_R$$

$$B_L = (B_R \text{ OR } k_{1R}) \text{ XOR } A_L$$

The left 32 bits of the output form B_L and the remaining form B_R .

3.1.6 FL⁻¹-function

FL⁻¹ function is called during both encryption and decryption processes. The input to the function is of 64 bits. The input data along with one of the subkeys is taken and then certain modifications are performed on it, finally returning a block of data also 64 bits in length. Let the input data block be X, while Y being the 64-bit long output block. k_1 is one of the subkeys previously created:

$$(B, k_1) \rightarrow A \Rightarrow (A_L \parallel B_R, k_{1L} \parallel k_{1R}) \rightarrow A_L \parallel A_R$$

Initially B is divided into two equal parts of 32 bits in length: B_L comprises the left bits of B, and B_R the right bits of B. Once this is over, two new blocks of 32-bits are calculated:

$$A_L = (B_R \text{ OR } k_{1R}) \text{ XOR } B_L$$

$$A_R = ((A_L \text{ AND } k_{1L}) \lll 1) \text{ XOR } B_R$$

The left 32 bits of the output form A_L and the remaining form A_R .

3.2 Camellia Decryption

Similar to any feistel cipher, encryption is carried out using the same algorithm but the order of the keys is reversed. This means that for a 128 bit decryption process, kw_1 is replaced by kw_3 and vice versa. Similarly, kw_2 is replaced with kw_4 , k_1 with k_{18} , k_2 with k_{17} and so on. The order of the keys used in FL and FL⁻¹ ($kl_{1..4}$) functions is reversed too, with kl_1 being replaced by kl_4 (and vice versa) and similarly, kl_2 with kl_3 .

3.3 Mutual Authentication

Mutual or two-way authentication is a protocol where each of the parties verify themselves at the same time about each other's identification and to exchange session keys. The most usual vulnerability of mutual authentication is Replay Attacks. Replay attacks are one in which the opponent genuinely copies a message and replays it later. To cope up with such attacks, we are able to use:

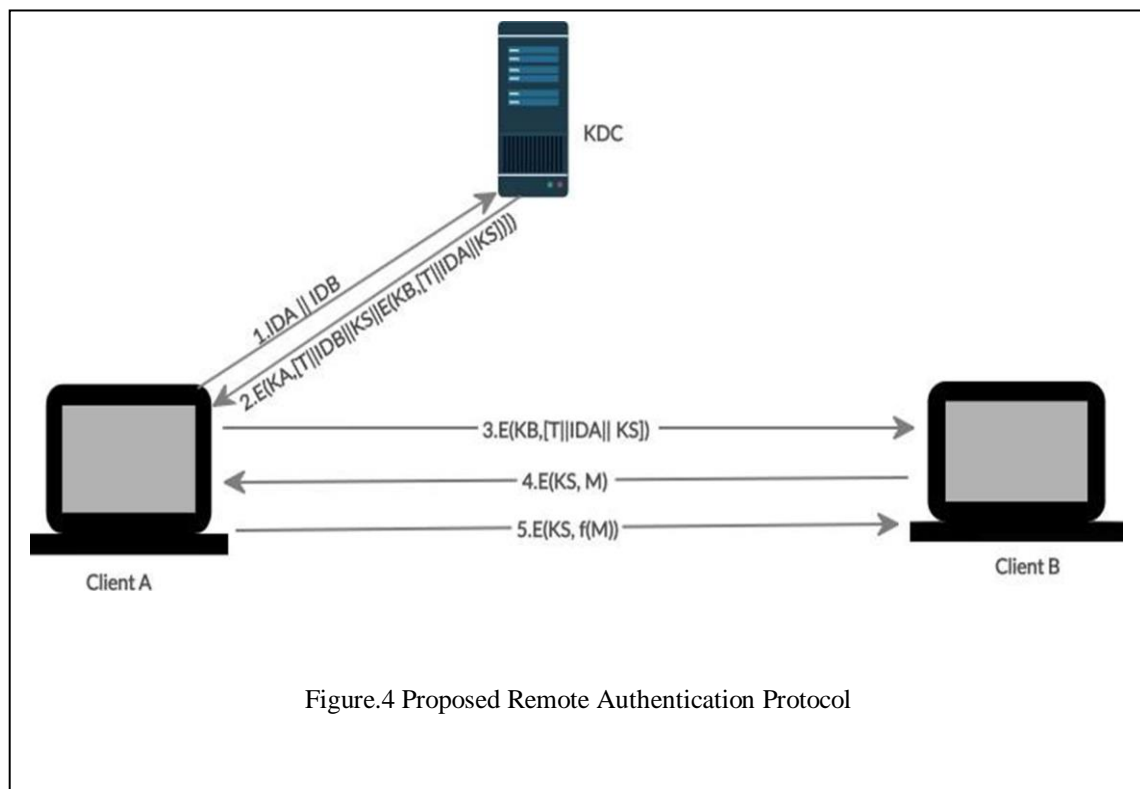
- **Timestamps:** A accepts a message as authentic only if the message has a timestamp.

However this could only be achieved if the clocks in the machine involved are synchronized .

- **Challenge/response:**A, waiting for a sparkling message from B, A first sends a nonce and requires that the response from B contain the perfect nonce value.

3.3.1 Mutual Authentication using Symmetric Key

1. ClientA \rightarrow Server : Identity(A) || Identity(B)
2. Server \rightarrow ClientA : $E(\text{Key}_A, [\text{Timestamp} || \text{Identity}(B) || \text{Key}_S || E(\text{Key}_B, [\text{Timestamp} || \text{Identity}(A) || \text{Key}_S])])$
3. ClientA \rightarrow ClientB : $E(\text{Key}_B, [\text{Timestamp} || \text{Identity}(A) || \text{Key}_S])$
4. ClientB \rightarrow ClientA : $E(\text{Key}_S, \text{Message})$
5. ClientA \rightarrow ClientB : $E(\text{Key}_S, f(\text{Message}))$



As shown in fig.4, keys Key_A and Key_B are shared between ClientA and the server i.e the key distribution centre. and ClientB and the Server, respectively. The ultimate goal of the protocol is to distribute a secret key Ks among A and B .

Timestamp makes sure ClientA and ClientB has the freshly generated session key. It allows both ClientA and ClientB to ensure that the key distribution is a fresh exchange. Client A and ClientB can verify this via checking the clock

In step 4, ClientB sends a message to ClientA. In step 5, ClientA sends again the function of the message to ClientB where the characteristic is known only to ClientA and ClientB. Thus, there's a handshake between them.

The encryption algorithm used for the above protocol is CAMELLIA.

3.3.2 Kerberos Version 5

Kerberos Version five has an extended ticket lifetime. It allows tickets to be renewed. In model 5, the ciphertext contains an encryption based identifier. In such cases, any encryption techniques can be used. Encryption keys too contain details about the type and their length. This allows the same key to be used in various algorithms and permitting the specification of one-of-a-kind variations on a given algorithm. Thus, It can take delivery of any symmetric-key algorithm.

Kerberos generally uses DES while it can accommodate any symmetric algorithm.

Kerberos 5 can be implemented using CAMELLIA encryption algorithm instead of DES.

1. Client \rightarrow AuthenticatingServer : Options||Identifier_C|| Realm_C||Identifier_{tgs}||Timestamp||N₁

2. AuthenticatingServer \rightarrow Client :

Realm_C||Identifier_C||Ticket_{tgs}||E(Key_{Client,tgs},[Key_{client,tgs}||Timestamp||N₁||Realm_{tgs}||Identifier_{tgs}])

$$\text{Ticket}_{tgs} = E(\text{Key}_{tgs}, [\text{Flags}||\text{K}_{client,tgs}||\text{Realm}_C||\text{Identifier}_C||\text{Address}_C||\text{Timestamp}])$$

(a) Authentication Server Exchange to obtain ticket-granting tickets

3. Client \rightarrow AuthenticatingServer : Options||Identifier_V||Timestamp||N₂||Ticket_{tgs}||Authenticator_C

4. TGS \rightarrow Client : Realm_C||Identifier_C||Ticket_V||E(Key_{Client,tgs},[K_{C,V}||Timestamp||N₂||Realm_C||Identifier_V])

$$\text{Ticket}_{tgs} = E(\text{Key}_{tgs}, [\text{Flags}||\text{Key}_{client,tgs}||\text{Realm}_C||\text{Identifier}_C||\text{Address}_C||\text{Timestamp}])$$

$$\text{Ticket}_V = E(\text{Key}_V, [\text{Flags}||\text{Key}_{C,V}||\text{Realm}_C||\text{Identifier}_C||\text{Address}_C||\text{Timestamp}])$$

$$\text{Authenticator}_C = E(\text{Key}_{Client,tgs}, [\text{Identifier}_C||\text{Realm}_C||\text{TimeStam}_1])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

5. Client \rightarrow Server : Options||Ticket_V||Authenticator_C

6. Server \rightarrow Client : E_{Key_{C,V}}[TimeStam₂||Subkey||Sequence#]

$$\text{Ticket}_V = E(\text{Key}_V, [\text{Flag}||\text{Key}_{C,V}||\text{Realm}_C||\text{Identifier}_C||\text{Address}_C||\text{Timestamp}])$$

$$\text{Authenticator}_C = E(\text{Key}_{C,V}, [\text{Identifier}_C||\text{Realm}_C||\text{TimeStam}_2||\text{Subkey}||\text{Sequence\#}])$$

(c) Client/Server Authentication Exchange to obtain Service

where,

Identifier_{tgs} = identification of the ticket granting server

Identifier_C = identifier of user on C

TGS = Ticket-granting server

Address_C = network address of C

V = server

Identifier_V = identifier of V

Key_V = secret encryption key shared by AS and V

Realm: describes the realm of user

Options: used to set flags in the tickets that are returned

N:Nonce: to ensure that replay attacks doesn't take place by checking if result is fresh

Subkey: The client's choice and demand per capita

Sequence number: An ambient and mathematically subtle optional field which gives the beginning number that can be used by a server for the messages for their further communications.

4. Experimental Analysis

4.1 Camellia

Encryption Pseudocode:

```
get plain-text from user
get key(128/192/256 bit) from user
convert the plain-text and key to byte array
setKey(true,key)           //initialize the key scheduling
processBlock(message,output) //encryption process
```

Decryption Pseudocode:

```
set the cipher text as input for decryption
set the key(128/192/256 bit)
convert the plain-text and key to byte array
setKey(false,key)          //initialize the key scheduling
processBlock(cipher,plaintext) //encryption process
```

setKey :

```
if key.length is equal to 128 do{
  set left keys value equal to key
  set right keys value as zero
}
if key.length is equal to 196 do{
  set left keys value equal to first 128 bits of key
  set right keys value as 64 bits of key and Masked 64bits of key
}
if key.length is equal to 256 do{
  set left keys value equal to first 128 bits of key
  set right keys value as last 128bits of key
}
```

Process Block :

```
For i=0 to 3 do{

  state[i] = state[i] XOR key[i]

}
camelliaF(state, subkey, 0);
camelliaF(state, subkey, 4);
camelliaF(state, subkey, 8);

camelliaFL(state, ke, 0); // inverse function after 6 rounds
camelliaF(state, subkey, 12);
camelliaF(state, subkey, 16);
camelliaF(state, subkey, 20);

camelliaFL(state, ke, 4); // inverse function after 6 rounds
camelliaF(state, subkey, 24);
camelliaF(state, subkey, 28);
camelliaF(state, subkey, 32);
```

Camellia round function [camelliaF()]

Variables d₁-d₈ are 8 bit unsigned integer

MASK(8) is equal to 0xff

>> performs left shift

<< performs left shift

^ performs XOR

```
d1 = x right shift 56;
d2 = (x right shift 48) & MASK(8);
d3 = (x right shift 40) & MASK(8);
d4 = (x right shift 32) & MASK(8);
d5 = (x right shift 24) & MASK(8);
d6 = (x right shift 16) & MASK(8);
d7 = (x right shift 8) & MASK(8);
d8 = x & MASK(8);
d1 = SBOX1[d1];
d2 = SBOX2[d2];
d3 = SBOX3[d3];
d4 = SBOX4[d4];
d5 = SBOX2[d5];
d6 = SBOX3[d6];
d7 = SBOX4[d7];
d8 = SBOX1[d8];
y1 = d1 ^ d3 ^ d4 ^ d6 ^ d7 ^ d8;
y2 = d1 ^ d2 ^ d4 ^ d5 ^ d7 ^ d8;
y3 = d1 ^ d2 ^ d3 ^ d5 ^ d6 ^ d8;
y4 = d2 ^ d3 ^ d4 ^ d5 ^ d6 ^ d7;
y5 = d1 ^ d2 ^ d6 ^ d7 ^ d8;
y6 = d2 ^ d3 ^ d5 ^ d7 ^ d8;
y7 = d3 ^ d4 ^ d5 ^ d6 ^ d8;
y8 = d1 ^ d4 ^ d5 ^ d6 ^ d7;
calculate cipher text by (t1 << 56) | (t2 << 48) | (t3 << 40) | (t4 << 32) | (t5 << 24) | (t6 << 16) | (t7 << 8) | t8
return cipher text
```

Camellia FL Function

```
For i=0 to 3 do{
Set state[i] equal to XOR of state[i] and cipher-text[i] from F function
}
return state
```

4.2 Remote User Authentication

ClientA :

```
Connect to the KDC
If connected do {
Send the details of identity A and B to KDC
Read the response from KDC
Decrypt the message using the public key of A
Store the decrypted message in byte arrays
get the session key from the decrypted message
set key = session key
}
Connect to ClientB
If connected do{
Send the encrypted message consisting the session key, timestamp and identity of A to B
Read the message from A
Decrypt the message using the public key of B
get the session key and set key=session key
}
```

```

else{
close session
}

```

KDC:

```

If connected to A do{
Set a randomly generated session key to a byte array
Set a timestamp
Encrypt the message consisting session key,identity of A and timestamp with public key of B
Encrypt the message consisting session key,identity of B, timestamp and the earlier encrypted message with
public key of A
Send the encrypted message to A
}
Close session

```

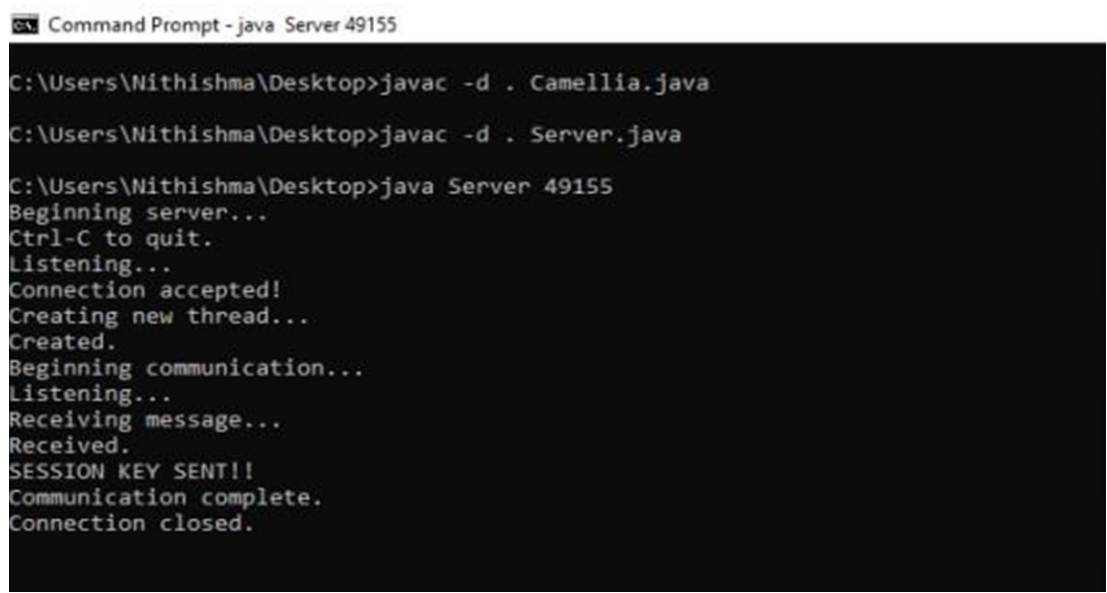
Client B :

```

If connected to A do{
Read the message from A
Decrypt the message using the public key of B
get the session key and set key=session key
get the user input message for authentication
send the message to A by encrypting it with session key
}
//A sends a function of the message(which is caesar cipher here)
Read the message from A and decrypt it
If(message from A == inverse of caesar cipher message)
{
The connection is authenticated
Start the communication
While(message != exit){
Send message to A
Get messages from A
}
}
Close session

```

SERVER



```

C:\Users\Nithishma\Desktop>javac -d . Camellia.java
C:\Users\Nithishma\Desktop>javac -d . Server.java
C:\Users\Nithishma\Desktop>java Server 49155
Beginning server...
Ctrl-C to quit.
Listening...
Connection accepted!
Creating new thread...
Created.
Beginning communication...
Listening...
Receiving message...
Received.
SESSION KEY SENT!!
Communication complete.
Connection closed.

```

Figure.5 Running the Server

ClientA

```
Command Prompt
C:\Users\Nithishma\Desktop>javac -d . ClientA.java
C:\Users\Nithishma\Desktop>java ClientA 127.0.0.1 65530 127.0.0.1 49155
Initialising User Authentication Protocol transaction...
    [Delta = 60]
    [Server At: 127.0.0.1:49155]
    [Target At: 127.0.0.1:65530]
Initialising connections...
Connected to Target!
Connected to Server!
Sending message...
Message sent.
Receiving message...
Received.
Sending {T, A, k<A,B>} to B...
Sent.
Receiving message...
Sending function of Message to UserB Nithishma!...
Sent.
AUTHENTICATED...
Enter 1 to start a conversation or press 0 to exit:
1
You can now begin a conversation with UserB.
Enter 'exit' to leave.
UserA : hey
UserB : hello
UserA : how are you??
UserB : im good,and you?
UserA : great..
UserB : little busy
UserA : oh thats ok,bye
C:\Users\Nithishma\Desktop>
```

Figure.6 Client A

ClientB

```
Command Prompt
C:\Users\Nithishma\Desktop>javac -d . ClientB.java
C:\Users\Nithishma\Desktop>java ClientB 65530
Beginning User Authentication- ClientB...
    [Delta: 60]
Ctrl-C to quit.
Listening...
Connection accepted!
Creating new thread...
Created.
Beginning communication...
Listening...
Receiving session key...
Received.
Enter your message: qazwsxedcrfvtgby
Sending Message for Authentication...
Sent.
Receiving message...
Decrypted.
AUTHENTICATED.....
UserA : hey
UserB : hello
UserA : how are you??
UserB : im good,and you?
UserA : great..
UserB : little busy
UserA : oh thats ok,bye
UserB : exit
C:\Users\Nithishma\Desktop>
```

Figure.7 Client B

From the figures 5,6 and 7 the process can be explained as:

1. A requests the KDC for the consultation key
2. KDC shares the consultation key and the encrypted message for B
3. A stocks the encrypted message to B
4. B decrypts and sends a message to A.
5. A replies to the feature of the message that B sent.
6. B tests if the messages are the same, if yes the verbal exchange begins.
7. The communication is closed when 'exit' is sent.

5. Strength of Camellia

-Use of 128/192/256 bit keys for encryption/decryption purposes

Assuming that partial the key space has to be combed, a solitary device carrying out one encryption per microsecond would take more than thousands of years to breakdown the cipher.

Table 1: Cryptanalysis times for various key sizes

Key Size (bits)	Number of Alternative Keys	Time Essential at 10^9 Decryptions/s	Time Essential at 10^{13} Decryptions/s
128	$2^{128} \approx 3.4 * 10^{38}$	2^{127} nanoseconds = $5.3 * 10^{21}$ Y	$5.3 * 10^{17}$ Y
196	$2^{192} \approx 6.3 * 10^{57}$	2^{191} nanoseconds = $9.8 * 10^{40}$ Y	$9.8 * 10^{36}$ Y
256	$2^{256} \approx 1.2 * 10^{77}$	2^{255} nanoseconds = $1.8 * 10^{60}$ Y	$1.8 * 10^{56}$ Y

-Avalanche effect : A minor variation in plaintext consequences in a identical excessive modification in the cipher text.

5.1 Why Camellia

Camellia is a feistel cipher similar to DES, SEED, Blowfish and various other encryption algorithms. We chose to use Camellia for the encryption and decryption processes in the aforementioned protocol since Camellia is a relatively lesser used algorithm even though it's almost as secure as AES. There have been claims of Camellia being equally as time efficient as AES and thus making its superiority to DES, axiomatic. We performed several tests to fact check these claims and we found that for smaller sizes of texts, Camellia is as fast as AES and in several cases, even faster. To compare performance, O-notation does not provide any interesting insights in the case of these encryption algorithms. Most of these algorithms normally work on a stable block size, and take roughly the identical time unconventionally of input, hence they are $O(1)$. In case of lengthier communications, you customarily get an $O(m)$ complexity, where m is the memorandum size, as you have $O(m)$ chunks of data to encode. As the strength of Camellia cipher is proven to be comparable to AES's , it is a potential alternative for other ciphers out there, especially other feistel ciphers such as DES, Blowfish etc. Cryptography Research and Evaluation Committee, CRYPTREC (Japan) has given AES and Camellia equal status. Camellia is still relatively lesser known than its competition and there hasn't been a lot of research on it. Due to these reasons, along with Camellia's clear advantages, we wanted to implement a mutual authentication scheme using Camelia to assess its performance and security.

6. Drawbacks of the developed model

6.1 Attacks on Camellia

Square Attack is based on a complex and randomized search scheme which selects localized square-shaped updates at random positions such and so that at each iteration the perturbation is situated approximately at the boundary of the feasible set. Camellia can also be attacked with square attack if it is processed by nine rounds 128-bits key Camellia without FL function or FL^{-1} functions layer and whitening is indisputably fragile with complexity of $2^{86.9}$ encryptions and 2^{66} data and 12-round 256-bit key Camellia without FL/ FL^{-1} function layer and whitening is breakable with the complexity of $2^{250.8}$ encryptions and 2^{66} data.

Recent developments have found that Camellia might be susceptible to certain side channel attacks, e.g., Cache trace attacks. Cache assaults are a class of side channel assaults that obtain secret info about the cipher from the deeds of the processor's cache memory. These attacks exploit the fact that a cache miss has different authority and mechanism when related to a cache hit.

6.2 Issues related to mutual authentication

Since to avoid replay attacks the use of timestamps and nonces becomes unavoidable. To effectively use timestamps, having a protocol that would maintain synchronization between the various processor clocks becomes of paramount importance. This protocol should be fault tolerant as well as able to handle network errors, and secure, to protect itself against attacks. If this protocol fails or falters, even for a single device, it gives rise to an opportunity for an attack. Thus even a temporary loss of synchronization could prove to be catastrophic for the system. This task becomes much more challenging when dealing with a large, scalable network. Finally, precise synchronization in distributed clocks can't be expected due to the unpredictable nature of network delays. This makes the network vulnerable to reduce replay attacks.

To prevent suppress-replay attacks, the involved parties must check the clock timings regularly against the Key Distribution Centre's clock. Another way is to depend on handshaking techniques using nonces. Using nonces as additional data for authentication, though increases security, adds overhead costs for communication. Another challenge with remote user authentication by mutual authentication is the requirement of a trusted server or Key Distribution center. If the KDC is compromised, the entire network and all devices connected to it are exposed.

7. COMPARISON OF CAMELLIA, AES, DES, BLOWFISH

Table 2: Properties of popular encryption algorithms

FACTORS	CAMELLIA	AES	DES	BLOWFISH
Developed in	2000	1998	1974	1993
Block size	128 bits	128 bits	64 bits	64 bits
Key length	128/196/256 bits	128/196/256 bits	64 (uses 56 bit)	32-448 bits
Cipher type	Symmetric	Symmetric	Symmetric	Symmetric
Structure	Feistel	Non-Feistel	Feistel	Feistel
Number of rounds	18/24	10/12/14	16	16
Number of S-box	4	1	8	4
Speed	Fast	Fast	Slow	Too fast
Security	Highly secure	Highly secure	Not enough secure	Secure
Open source	No	Yes	Yes	Yes
Encryption Time	High	High	High	High
Decryption Time	High	High	Medium	Low
Best known attacks	Square attacks	7/8/9 rounds	EFF DEScracking machine	4 rounds, SWEET32 attack
PROCESSING TIME FOR THE DEVELOPED MODEL	5.611 sec (or) 5611 ms	7.012s ec (or) 7012 ms	12.367sec (or) 12367 ms	4.44sec (or) 4440 ms

In the table above, we've compared some of the more popular encryption algorithms based on a variety of factors. AES is the only non-feistel cipher that we've included and only Camellia is non-open source. Camellia and AES are comparatively more secure and Blowfish seems to be the most efficient for the proposed protocol.

The following figures depict the processing time for the developed model with different encryption algorithms. Figure 8 shows the processing speed of Camellia Encryption algorithm. Figure 9 shows the processing time for AES algorithm. Figure 10 is about the processing time for DES algorithm and Figure 11 shows the processing speed of the model using blowfish algorithm.

```
C:\Users\Nithishma\Desktop>java ClientA 127.0.0.1 65530 127.0.0.1 49155
Initialising User Authentication Protocol transaction...
    [Delta = 60]
    [Server At: 127.0.0.1:49155]
    [Target At: 127.0.0.1:65530]
Initialising connections...
Connected to Target!
Connected to Server!
Sending message...
Message sent.
Receiving message...
Received.
Sending {T, A, k<A,B>} to B...
Sent.
Receiving message...
Sending function of Message to UserB Nithishma!...
Sent.
AUTHENTICATED...
Total processing time of CAMELLIA in the developed model is : 5611
```

Figure.8 Calculating Camellia's processing speed

```
C:\Users\Nithishma\Desktop>java ClientA 127.0.0.1 65532 127.0.0.1 49157
Initialising User Authentication Protocol transaction...
    [Delta = 60]
    [Server At: 127.0.0.1:49157]
    [Target At: 127.0.0.1:65532]
Initialising connections...
Connected to Target!
Connected to Server!
Sending message...
Message sent.
Receiving message...
Received.
Sending {T, A, k<A,B>} to B...
Sent.
Receiving message...
Sending function of Message to UserB Nithishma!...
Sent.
Total processing time of AES in the developed model is : 7012
```

Figure.9 Calculating AES's processing speed

```
C:\Users\Nithishma\Desktop>java ClientA 127.0.0.1 65531 127.0.0.1 49156
Initialising User Authentication Protocol transaction...
    [Delta = 60]
    [Server At: 127.0.0.1:49156]
    [Target At: 127.0.0.1:65531]
Initialising connections...
Connected to Target!
Connected to Server!
Sending message...
Message sent.
Receiving message...
Received.
Sending {T, A, k<A,B>} to B...
Sent.
Receiving message...
Sending function of Message to Nithi007...
Sent.
Total processing time of DES in the developed model is : 12367
```

Figure.10 Calculating DES's processing speed


```

C:\Users\Nithishma\Desktop>java ClientA 127.0.0.1 65444 127.0.0.1 49160
Initialising User Authentication Protocol transaction...
[Delta = 60]
[Server At: 127.0.0.1:49160]
[Target At: 127.0.0.1:65444]
Initialising connections...
Connected to Target!
Connected to Server!
Sending message...
Message sent.
Receiving message...
Received.
Sending {T, A, k<A,B>} to B...
Sent.
Receiving message...
Sending function of Message to Nithi007...
Sent.
Total processing time of BLOWFISH in the developed model is : 4440

```

Figure.11 Calculating Blowfish's processing speed

7.1 COMPARISON CHARTS

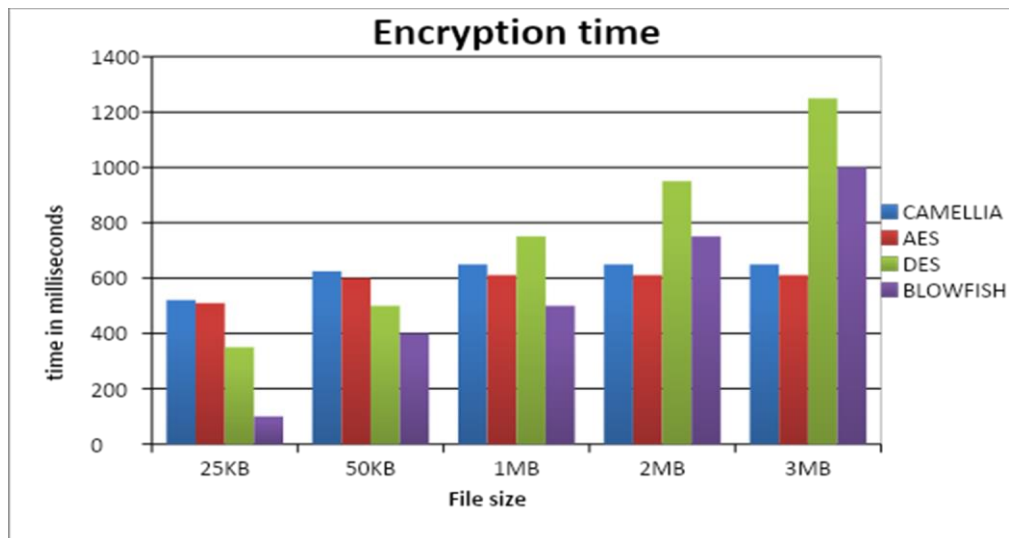


Figure.12 Encryption time chart for various ciphers

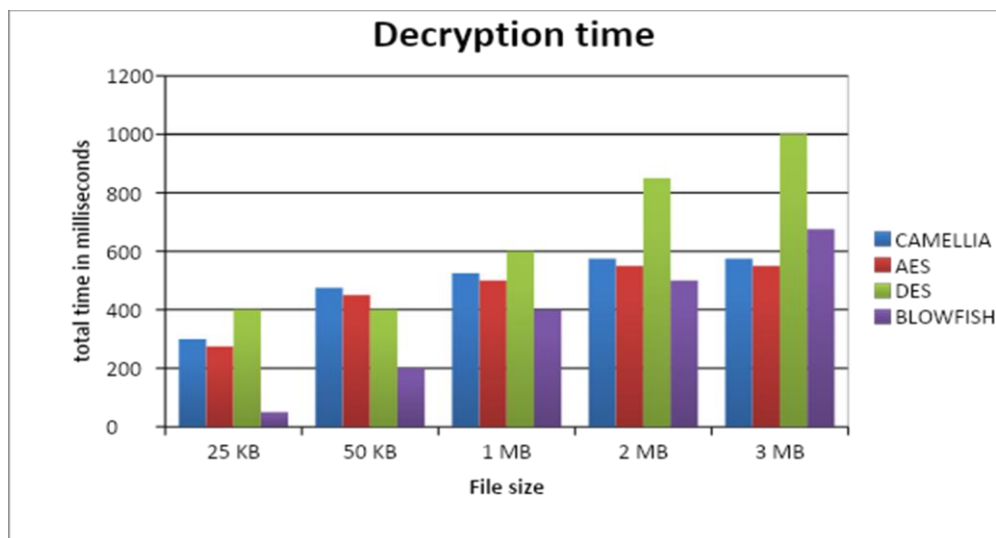


Figure.13 Decryption times for various ciphers

From the given comparisons, i.e. Figure 12 depicts comparison of the Encryption times of various algorithms while Figure 13 shows the Decryption time for those algorithms, we can see that for the encryption process Blowfish is the fastest for smaller data sizes. DES turns out to be the slowest of the four. Camellia and AES have comparable performance, with AES being just slightly more efficient. Similar conclusions can be drawn for their decryption times as well.

8. Conclusion

Camellia has no linear or differential characteristics of probability greater than 2^{-128} . The design of Camellia is such that it offers great security over other advanced cryptanalytic attacks including boomerang attacks, higher order differential attacks, truncated differential attacks, interpolation attacks, related-key attacks, and slide attacks. Thus it's safe to say that the Camellia cipher is quite secure. However, even with the promise of such a level of security, use of Camellia for User authentication or Remote user authentication is an unpopular choice. Our project tries to explore the possible reasons for this.

We have developed an environment which provides mutual authentication over a network and uses Camellia encryption for the steady channel communications. We furnished timestamps such that replay attacks can't take place. In case the network is attacked, the users are notified and further communication is closed. The identical surroundings were also tested using other encryption algorithms such as AES, DES and BLOWFISH. We have provided the processing times of every algorithm and according to our tests, blowfish appears to be the fastest followed by Camellia, AES and DES respectively. A comparative study of the algorithms has been made, keeping in mind the elements such as vulnerability to attacks, energy and speed. After analyzing the results, we found that Camellia and AES are the most suitable for the developed verbal exchange channel. However, if one was to pick between the two, they'd probably still choose AES, owing to its popularity, however, we have also fairly established that it is not because Camellia is far less efficient or far less secure than AES.

REFERENCES

1. Aoki K. et al. (2001) Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis. In: Stinson D.R., Tavares S. (eds) *Selected Areas in Cryptography*. SAC 2000. Lecture Notes in Computer Science, vol 2012. Springer, Berlin, Heidelberg.
2. Matsui, M. & Nakajima, J. & Moriai, Shiho. (2004). A Description of the Camellia Encryption Algorithm, RFC3713
3. Wang, Yan & Wang, Nihong. (2012). Research on Time-Varying Camellia Encryption Algorithm. 162. 823-828. 10.1007/978-3-642-29455-6_110.
4. Kanda M., Matsumoto T. (2002) Security of Camellia against Truncated Differential Cryptanalysis. In: Matsui M. (eds) *Fast Software Encryption*. FSE 2001. Lecture Notes in Computer Science, vol 2355. Springer, Berlin, Heidelberg.
5. Srivastava K., Awasthi A.K., Mittal R.C. (2013) A Review on Remote User Authentication Schemes Using Smart Cards. In: Singh K., Awasthi A.K. (eds) *Quality, Reliability, Security and Robustness in Heterogeneous Networks*. QShine 2013. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 115. Springer, Berlin, Heidelberg
6. Aoki, Ichikawa, Kanda, Matsui, Moriai, Nakajima, Tokita, "Specification of Camellia — a 128-bit Block Cipher," <http://info.isl.ntt.co.jp/camellia/>, 2000.
7. Zhao, Xin-jie & Wang, Tao & Zheng, Yuanyuan. (2009). Cache Timing Attacks on Camellia Block Cipher.. IACR Cryptology ePrint Archive. 2009. 354.
8. Usman, Muhammad & Khan, Shujaat. (2017). SIT: A Lightweight Encryption Algorithm for Secure Internet of Things. *International Journal of Advanced Computer Science and Applications*. 8. 10.14569/IJACSA.2017.080151.
9. Cica, Zoran. (2016). Pipelined implementation of Camellia encryption algorithm. 1-4. 10.1109/TELFOR.2016.7818785.
10. Mahamat, Youssouf et al. "Comparative Study Of AES, Blowfish, CAST-128 And DES Encryption Algorithm." *IOSR Journal of Engineering* 06 (2016): 01-07.
11. H. Cheng and H. M. Heys, "Compact Hardware Implementation of the Block Cipher Camellia with Concurrent Error Detection," 2007 Canadian Conference on Electrical and Computer Engineering, Vancouver, BC, 2007, pp. 1129-1132, doi: 10.1109/CCECE.2007.287.
12. B. J. Mohd and T. Hayajneh, "Lightweight Block Ciphers for IoT: Energy Optimization and Survivability Techniques," in *IEEE Access*, vol. 6, pp. 35966-35978, 2018, doi: 10.1109/ACCESS.2018.2848586.
13. Y. Deng, T. Xie, H. Shi and J. Gong, "Research on the F-function of Camellia," 2011 International Conference on Electrical and Control Engineering, Yichang, 2011, pp. 1232-1235, doi: 10.1109/ICECENG.2011.6057840.
14. Moriai, Shiho, Akihiro Kato and Masayuki Kanda. "Addition of Camellia Cipher Suites to Transport Layer Security (TLS)." RFC 4132 (2005): 1-7.
15. P. Yalla and J. Kaps, "Compact FPGA implementation of Camellia," 2009 International Conference on Field Programmable Logic and Applications, Prague, 2009, pp. 658-661, doi: 10.1109/FPL.2009.5272349.
16. <http://www.crypto-it.net/eng/>

17. <https://1pdf.net/>
18. Rishabh Poddar, Amit Datta, and Chester Rebeiro. 2011. A cache trace attack on CAMELLIA. In Proceedings of the First international conference on Security aspects in information technology (InfoSecHiComNet'11). Springer-Verlag, Berlin, Heidelberg, 144–156.
19. Aoki K. et al. (2001) *Camellia*: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis. In: Stinson D.R., Tavares S. (eds) Selected Areas in Cryptography. SAC 2000. Lecture Notes in Computer Science, vol 2012. Springer, Berlin, Heidelberg.
20. Shukla, Sanjay Singh. "Chapter 50 Modeling and Verification of Inter Realm Authentication in Kerberos Using Symbolic Model Verifier" , Springer Science and Business Media LLC, 2011
21. <http://www.cs.haifa.ac.il/~orrd/LC17/paper60.pdf>
22. Nazeh Abdul Wahid MD, Ali A, Esparham B, Marwan MD (2018) A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention. J Comp Sci Appl Inform Technol. 3(2): 1-7.