Report Master Colloquium A

Giorgio Sidari

-Matriculation number: 108219

Uncertainity of Online Transaction Systems

Summary

Uncertainity of Online Transaction Systems

Introduction

Common Attacks and Technologies in Use

Common attacks on payment transactions include:

1. SQL Injection: Attackers use malicious queries to manipulate or extract sensitive data from databases, such as user credentials or transaction amounts.
2. Cross-Site Scripting (XSS): Exploiting XSS vulnerabilities, attackers impersonate legitimate users to carry out fraudulent transactions.
3. Distributed Guessing Attacks: These exploit inconsistencies in payment gateway security implementations to steal card data.
4. Man-in-the-Middle Attacks: Data transmitted over insecure channels is intercepted and manipulated.
5. Weak Storage Mechanisms: Inadequate encryption of stored data, such as credit card details, creates vulnerabilities. Systems relying on outdated encryption or plaintext storage are particularly at risk.

Despite these vulnerabilities, modern online payment systems employ sophisticated technologies to enhance security:

- Robust Cryptographic Algorithms:
    - *AES (Advanced Encryption Standard)*: A symmetric encryption method offering high resistance to brute-force attacks.
    - *RSA (Rivest-Shamir-Adleman)*: An asymmetric encryption cornerstone, using public and private key pairs.
    - *Hash Functions (e.g., SHA-512)*: Ensure data integrity by generating unique, irreversible outputs.
- EMV Technology:
    - Chip-based cards (EMV: Europay, MasterCard, Visa) enhance security with tamper-resistant chips storing encrypted information.
- 3D Secure Protocols:
    - Protocols like Verified by Visa and MasterCard SecureCode introduce additional authentication steps, reducing unauthorized transactions.
- Steganography and Visual Cryptography:
    - Steganography conceals sensitive information within benign content.
    - Visual cryptography splits data into multiple shares, ensuring no single entity holds complete information.

AES encryption

AES is an iterative symmetric block cipher designed for electronic data security. Its simplicity lies in its use of bytes, making implementation straightforward. AES processes 128-bit data blocks through transformation rounds, producing ciphertext as output. The key length (128, 192, or 256 bits) determines the number of rounds (10, 12, or 14, respectively).

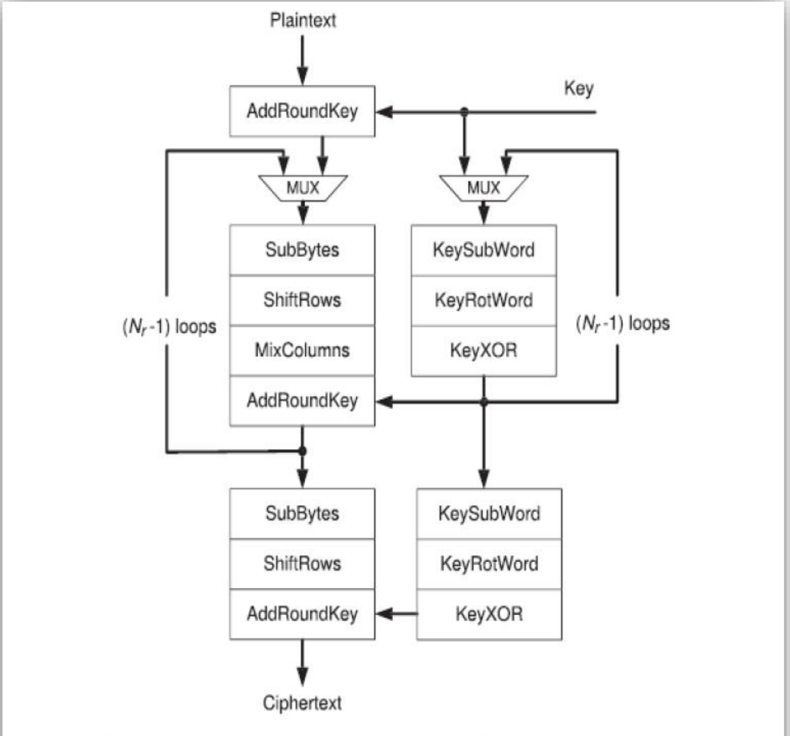


Fig. 1. Block diagram of AES encryption.

The four AES steps are listed below:

- **Sub bytes**: It is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table (S box). This S-box which is invertible and is constructed by first taking the multiplicative inverse in the *Galois Field* (2^8) with irreducible polynomial m(x) = x^8 + x^4 + x^3 + x + 1. The element {00} is mapped to itself. Then affine transformation is applied (over GF (2)).

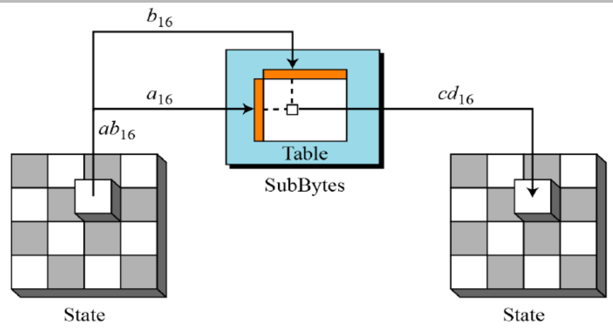


Fig 1.1: Substitution

- **Shift Rows**: In this transformation, the first row of the state array kept as it is. But the bytes in the second, third, and forth rows are cyclically shifted by one, two, and three bytes to the left, respectively.

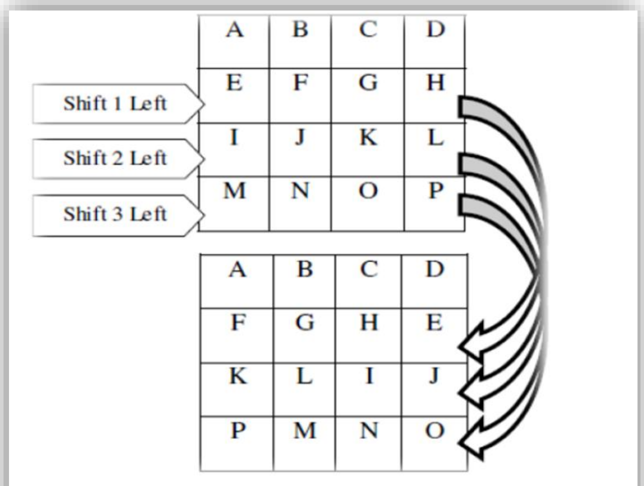


Fig 1.2: Shifting Rows

- **Mix** Columns: here columns of the state are considered as polynomials over GF (2^8) and multiplied by modulo x^4 + 1 with a fixed polynomial c(x), given by:
  c(x)={03}x^3 + {01}x^2 + {01}x + {02}.

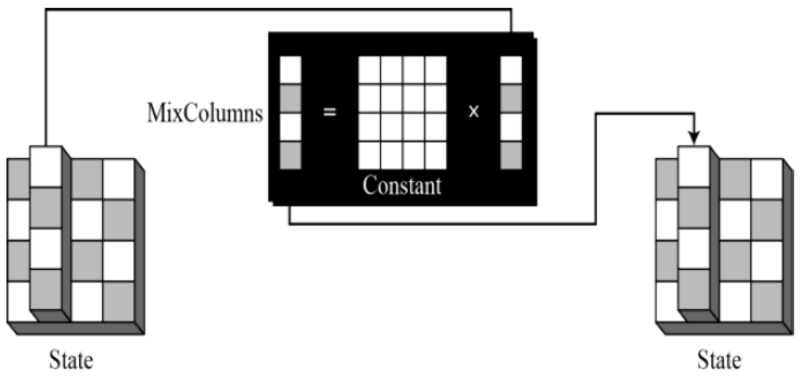


Fig 1.3: Mixing

- **Add Roundkey**: Using a bitwise exclusive-or (*XOR*) operation, a round key is added to the state array. This round keys are computed in the key expansion process. If Round keys are calculated for each data block, then it is called AES with online key expansion. On the other hand, in most of the applications, the encryption keys do not change as frequently as data. As a result, before the encryption process round keys can be calculated and kept constant for a period of time in local registers or memory.
  This computation of round key is called AES with offline key expansion.

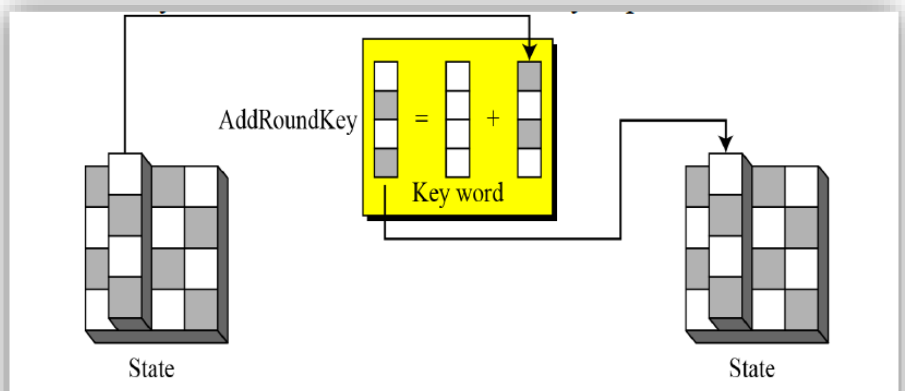


Fig 1.4: Adding Key

Similarly, there are three steps in each key expansion round:

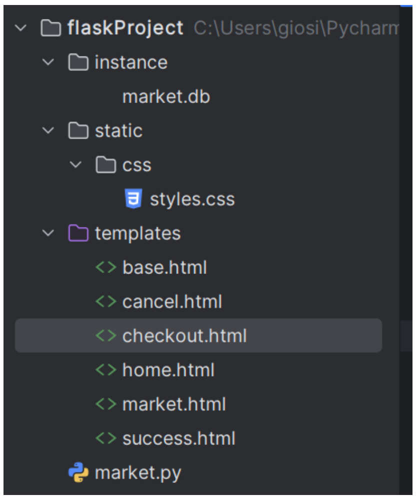
- **KeySubWord**: it takes a four byte input word and output word is produced by substituting each byte in the input to another byte according to the S-box.
- **KeyRotWord**: The second step of key expansion function is KeyRotWord which takes a word [a3, a2, a1, a0], performs a cyclic permutation, and returns the word [a2, a1, a0, a3] as output.
- **KeyXOR**: In key XOR function of key expansion, Every word w[i] is equal to the XOR of the previous word, w [i-1], and the word Nk positions earlier, w [i –Nk]. Nk equals 4, 6 or 8 for the key lengths of 128, 192 or 256 bits, respectively.

Prototype with **Stripe**

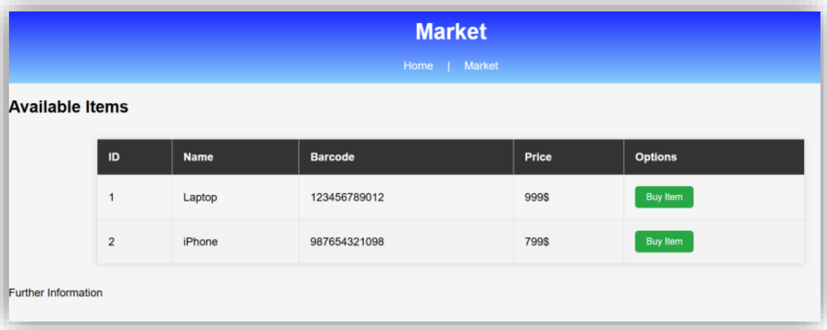
A prototype website was developed using REST (***RE****presentational* ***S****tate* ***T****ransfer*) APIs and the Stripe library for test online transactions. Stripe, a widely used payment platform, incorporates AES encryption.

- **Frontend**: Scripts (e.g., styles.css, base.html) ensure consistent page layouts.

- **Backend**: Implemented in Python with Flask (micro Web framework). Scripts like *market.py* manage the database, market page, and checkout session. Transactions redirect users to test checkout pages, confirming successful or failed payments via predefined responses.



```
flaskProject C:\Users\giosi\Pycharm
  instance
      market.db
  static
    css
        styles.css
  templates
      base.html
      cancel.html
      checkout.html
      home.html
      market.html
      success.html
  market.py
```



Market

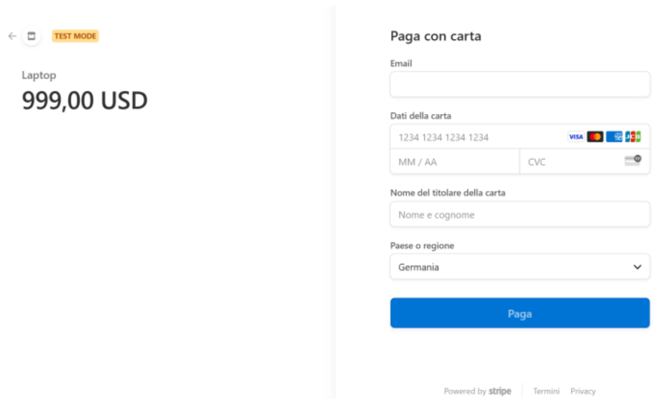
Home | Market

Available Items

ID	Name	Barcode	Price	Options
1	Laptop	123456789012	999$	Buy Item
2	iPhone	987654321098	799$	Buy Item

Further Information

After pressing "buy item" button, the page will redirect to the checkout page.



TEST MODE

Laptop

999,00 USD

Paga con carta

Email

Dati della carta

1234 1234 1234 1234

MM / AA

CVC

Nome del titolare della carta

Nome e cognome

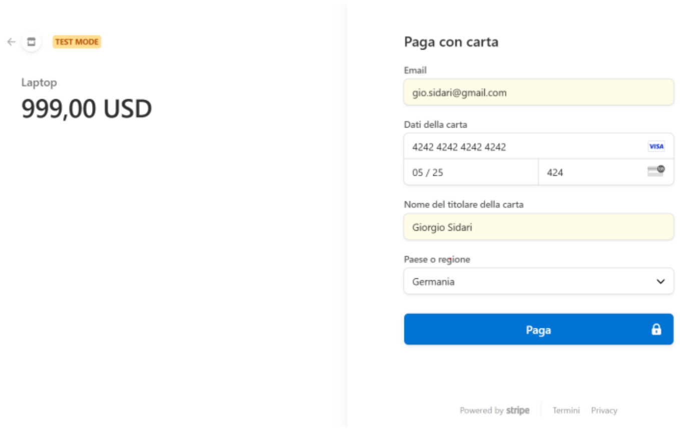
Paese o regione

Germania

Paga

Powered by stripe | Termini Privacy

Here it is the checkout in test mode, it is possible to add one of the test prototype which are available in Stripe documentation.



TEST MODE

Laptop

999,00 USD

Paga con carta

Email

gio.sidari@gmail.com

Dati della carta

4242 4242 4242 4242

05 / 25

424

Nome del titolare della carta

Giorgio Sidari

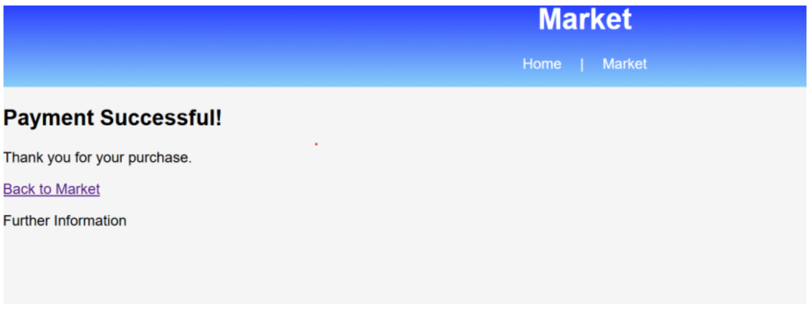
Paese o regione

Germania

Paga

Powered by stripe | Termini Privacy

After clicking on Buy ("Paga" in Italian) the session will move to page *success.html*, if credentials are correct, to *cancel.html* if the payment was not completed



Market

Home | Market

Payment Successful!

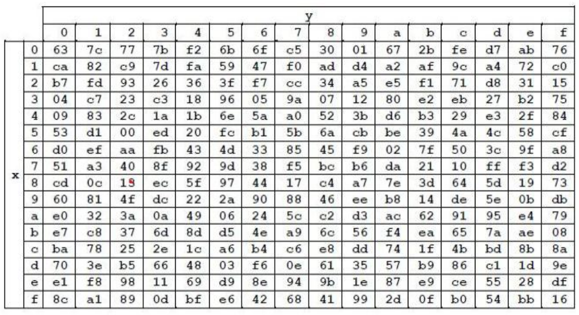
Thank you for your purchase.

Back to Market

Further Information

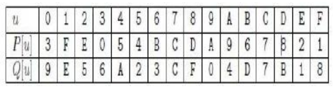
AES OPTIMIZATION

The proposed system compares AES encryption with two different S-Boxes. S-box with 256 byte lookup table (**Rijndael S-Box**) and AES with 16 byte S-Box (**Anubis S-Box**). Anubis encrypts blocks of 128 bits, which are internally represented as 16 bytes arranged in a 4-by-4 matrix.



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
x	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Fig. 2. Rijndael S-box(256 byte)



u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$P[u]$	3	F	E	0	5	4	B	C	D	A	9	6	7	B	2	1
$Q[u]$	9	E	5	6	A	2	3	C	F	0	4	D	7	B	1	8

Fig. 3. AnuBis S-box(16 byte)

The most important differences between Anubis and Rijndael are the following:

- **Involutional structure**: In Anubis, all steps are involutions. This implies that the implementation of the decryption operation can be the same as the encryption operation, except for a change in the key schedule(while in Rijndael encrypting and decrypting shift rows step can bring to different values). This should in principle reduce the code size or area in software, respectively hardware applications that implement both encryption and decryption.
- **Different S-box**: Anubis S-box is generated in a pseudo-random way. The advantage of this method is that providing a simple mathematical description seems more difficult. The polynomial expansion of the S-box is certainly more involved. The disadvantages are the suboptimal differential and linear properties, and possibly a more complex hardware implementation.

- **more complex key scheduling**: From the lookup table of Anubis, element u represent input byte. In AES with Anubis S-box, during SubByte transformation, the first four bytes of input (represented by **u**) are substituted by corresponding **p[u]** and last four bytes(represented by **u**) are substituted by corresponding **Q[u].**
  The advantage is the improved resistance against key based attacks, in particular the shortcuts for long keys. The 16 bytes S-box uses less memory space compared with 256 bytes S-box. It also reduces the hardware utilization and power consumption and reduces the execution time for encryption process.

Introduction to RSA

RSA is an asymmetric algorithm used in cryptography (*Asymmetric encryption is the process of using a public key from a public/private key pair to encrypt plaintext, and then using the corresponding private key to decrypt the ciphertext*), mainly based on the definition of totient and on modulus operations.

R.S.A. is named after its three inventors: Ronald Rivest, Adi Shamir e Leonard Adleman

The functionality of the RSA algorithm is based on aspects of number theory involving prime numbers and modulus operations. **Euler's totient** is an important part of the RSA key generation algorithm. Euler's Totient or simply the totient is a term that denotes all the integers smaller than **n** that have no common factors with **n**. Another way of stating this is how many numbers smaller than **n** are co-prime with **n** (*note: two numbers are co-prime if they have no common factors*). For example, if **n = 8** then you are asking how many numbers smaller than 8 have no common factors with it. The answer would be 3, 5, and 7. 1 is also included as a special case, thus the totient of 8 would be 4.

So, how can I calculate the totient of a prime number?

If, for example, **n = 7** then all positive integers smaller than **n** are co-prime to **n**. So, the totient of 7 is 6. Put more generally, the totient of any prime number **p** is **p-1**. Moreover, if **k=m*n** (both **m** and **n** prime numbers), the totient of that **k** is **(m-1)(n-1)**.

In addition, the modulus operation is a key mathematical component of the RSA algorithm. Modulus operations are important in cryptography, particularly in RSA. To use them, given a number **A** to which we would like to calculate the modulus operation **N**, $A \bmod (N)$, simply divide **A** by **N** and return the remainder.

After these clarifications, we can go in deep with RSA algorithm.

RSA algorithm

Step 1: Generate two large random primes, **p** and **q**, of approximately equal size such that their product $n = p*q$ is of the required bit length (such as 2048 bits, 4096 bits, etc.)

Step 2: Let $n = p*q$ and let $m = (p-1)*(q-1)$

Step 3: Choose a small number **e**, co-prime to **m**.

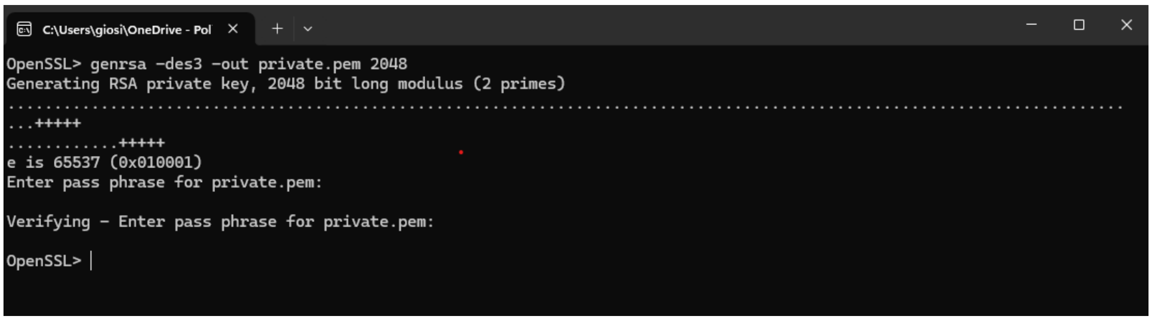
Step 4: Find **d**, such that $d*e \bmod m \equiv 1$. The modulus operator, if you are unfamiliar with it means to divide **a** number and select the remainder. So, we are looking for a number **d**, then when multiplied by **e** and divided by **m** gives 1 as a remainder. And that is it, now just publish **e** and **n** as the public key. Keep **d** and **n** as the *secret key*.

How can I generate a public and private key pair using RSA?

One of the most popular commands in SSL to create, convert, manage the SSL Certificates is OpenSSL.

You can generate a public and private RSA key pair like this:

1) start OpenSSL and write this line:
   genrsa -des3 -out privateKeyTest.pem 2048

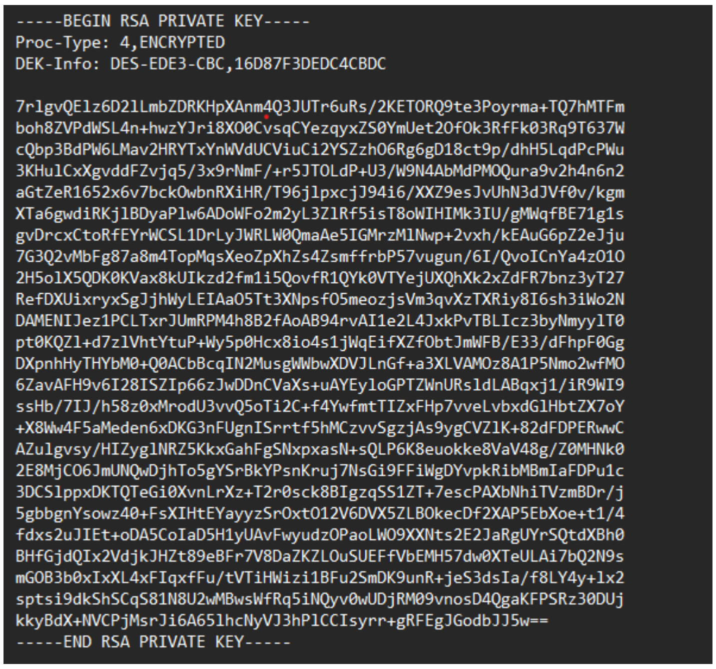


```
OpenSSL> genrsa -des3 -out private.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................................................................
...+++++
...........+++++
e is 65537 (0x010001)
Enter pass phrase for private.pem:

Verifying - Enter pass phrase for private.pem:

OpenSSL> |
```

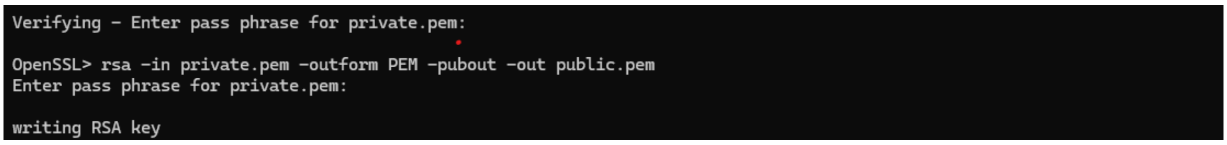
Now, opening the file private.pem, the output should be similar to the following:



```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,16D87F3DEDC4CBDC

7rlgvQElz6D21LmbZDRKHpXAnm4Q3JUTr6uRs/2KETORQ9te3Poyrma+TQ7hMTFm
boh8ZVPdWSL4n+hwzYJri8XO0CvsqCYezqyxZS0YmUet2OfOk3RfFk03Rq9T637W
cQbp3BdPW6LMav2HRYTxYnWVdUCViuCi2YSZzhO6Rg6gD18ct9p/dhH5LqdPcPWu
3KHulCxXgvddFZvjq5/3x9rNmF/+r5JTOLdP+U3/W9N4AbMdPMOQura9v2h4n6n2
aGtZeR1652x6v7bckOwbnRXiHR/T96jlpxcjJ94i6/XXZ9esJvUhN3dJVf0v/kgm
XTa6gwdiRKjlBDyaPlw6ADoWFo2m2yL3ZlRf5isT8oWIHIMk3IU/gMWqfBE71g1s
gvDrcxCtoRfEYrWCSL1DrLyJWRLW0QmaAe5IGMrzMlNwp+2vxh/kEAuG6pZ2eJju
7G3Q2vMbFg87a8m4TopMqsXeoZpXhZs4ZsmffrbP57vugun/6I/QvoICnYa4zO1O
2H5olX5QDK0KVax8kUIkzd2fm1i5QovfR1QYk0VTYejUXQhXk2xZdFR7bnz3yT27
RefDXUixryxSgJjhWyLEIAaO5Tt3XNpsfO5meozjsVm3qvXzTXRiy8I6sh3iWo2N
DAMENIJez1PCLTxrJUmRPM4h8B2fAoAB94rvAI1e2L4JxkPvTBLIcz3byNmyylT0
pt0KQZl+d7zlVhtYtuP+Wy5p0Hcx8io4s1jWqEifXZfObtJmWFB/E33/dFhpF0Gg
DXpnhHyTHYbM0+Q0ACbBcqIN2MusgWWbwXDVJLnGf+a3XLVAMOz8A1P5Nmo2wfMO
6ZavAFH9v6I28ISZIp66zJwDDnCVaXs+uAYEyloGPTZWnURsldLABqxj1/iR9WI9
ssHb/7IJ/h58z0xMrodU3vvQ5oTi2C+f4YwfmtTIZxFHp7vveLvbxdGlHbtZX7oY
+X8Ww4F5aMeden6xDKG3nFUgnISrrtf5hMCzvvSgzjAs9ygCVZlK+82dFDPERwwC
AZulgvsy/HIZyglNRZ5KkxGahFgSNxpxasN+sQLP6K8euokke8VaV48g/Z0MHNk0
2E8MjCO6JmUNQwDjhTo5gYSrBkYPsnKruj7NsGi9FFiWgDYvpkRibMBmIaFDPu1c
3DCSlppxDKTQTeGi0XvnLrXz+T2r0sck8BIgzqSS1ZT+7escPAXbNhiTVzmBDr/j
5gbbgnYsowz40+FsXIHtEYayyzSrOxtO12V6DVX5ZLBOkecDf2XAP5EbXoe+t1/4
fdxs2uJIEt+oDA5CoIaD5H1yUAvFwyudzOPaoLWO9XXNts2E2JaRgUYrSQtdXBh0
BHfGjdQIx2VdjkJHZt89eBFr7V8DaZKZLOuSUEFfVbEMH57dw0XTeULAi7bQ2N9s
mGOB3b0xIxXL4xFIqxfFu/tVTiHWizi1BFu2SmDK9unR+jeS3dsIa/f8LY4y+lx2
sptsi9dkShSCqS81N8U2wMBwsWfRq5iNQyv0wUDjRM09vnosD4QgaKFPSRz30DUj
kkyBdX+NVCPjMsrJi6A65lhcNyVJ3hPlCCIsyrr+gRFEgJGodbJJ5w==
-----END RSA PRIVATE KEY-----
```

2) The previous command generates a 2048-bit RSA key pair, encrypts them with a password you provide, and writes them to a file. You need to next extract the public key file. You will use this, for instance, on your web server to encrypt content so that it can only be read with the private key.
3) Export the RSA Public Key to a File
    a. type this command:
        ➢ *rsa -in private.pem -outform PEM -pubout -out public.pem*

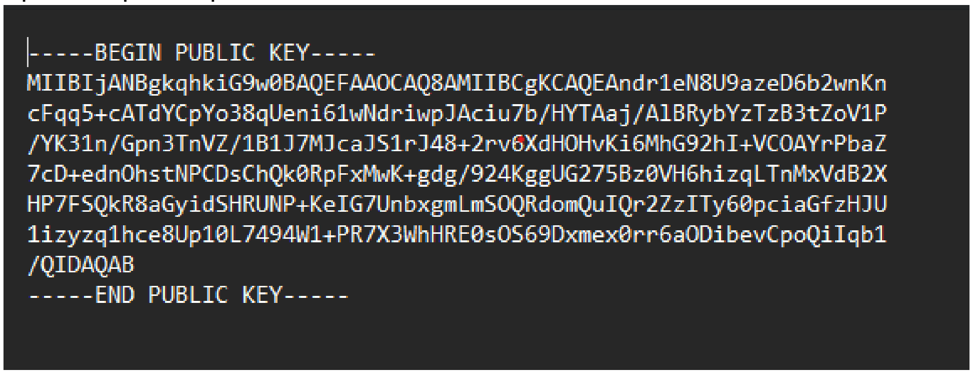


```
Verifying - Enter pass phrase for private.pem:

OpenSSL> rsa -in private.pem -outform PEM -pubout -out public.pem
Enter pass phrase for private.pem:

writing RSA key
```

4) Next open the public.pem and ensure that it starts with a: -----BEGIN PUBLIC KEY-----

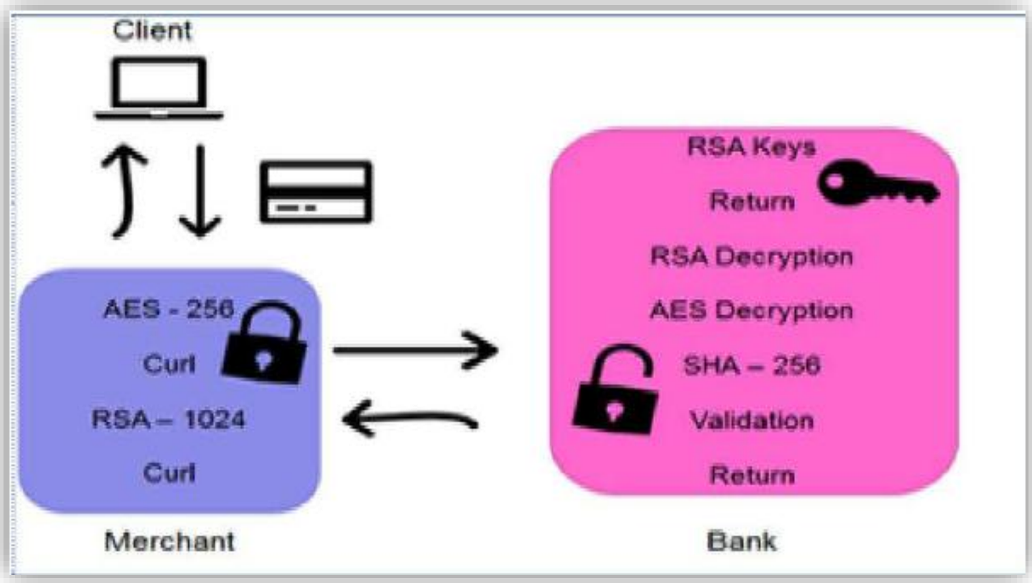


```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAndr1eN8U9azeD6b2wnKn
cFqq5+cATdYCpYo38qUeni61wNdriwpJAciu7b/HYTAaj/AlBRybYzTzB3tZoV1P
/YK31n/Gpn3TnVZ/1B1J7MJcaJS1rJ48+2rv6XdHOHvKi6MhG92hI+VCOAYrPbaZ
7cD+ednOhstNPCDsChQk0RpFxMwK+gdg/924KggUG275Bz0VH6hizqLTnMxVdB2X
HP7FSQkR8aGyidSHRUNP+KeIG7UnbxgmLmSOQRdomQuIQr2ZzITy60pciaGfzHJU
1izyzq1hce8Up10L7494W1+PR7X3WhHRE0sOS69Dxmex0rr6aODibevCpoQiIqb1
/QIDAQAB
-----END PUBLIC KEY-----
```

Optimal method to enhance security

Now that we have a clear idea of how AES and RSA work, the following step is to introduce a new approach for enhancing security of online payment systems. We will do three important modifications to systems that exist already:

- Designing a unified architecture to eradicate the distributed guessing attack.
- Restricting the limit on invalid entry of a card's details for a single transaction.
- Encrypting the communication and storing the encrypted form of credit/debit card details in the database.



Nowadays different sites have different input fields in their payment processing page like some have Card no., Exp, CVV; and some have an additional field as card holder name; some sites don't have CVV, instead they have 3D secure pin. These differences in the input fields and number of fields make it easy for the hacker to steal data. So it is recommended for all the merchant sites to use the same architecture for the payment process. Using a unified architecture can: eradicate distributed guessing attack to most extent, be implemented with the latest security algorithms and be easily updated later in the future.

Steps

1. User enters his/her credit card details on the merchant site. The data is sent to merchant server;
2. AES key and AES initialization vector for 256 bits are generated;
3. Generated keys are used to encrypt data acquired from the merchant site with AES and Chain Block Cyphering (CBC);
4. After the encryption, merchant server sends a request to bank server for an RSA 1024 bits public key;
5. AES key and AES vector are encrypted using RSA;
6. The encrypted message is sent to the bank server using cURL;
7. RSA decryption;
8. AES decryption;
9. Except the card number, remaining card details are encrypted using sha512, because in the bank database are encrypted using it
10. Encrypted data is compared with the one present in the database for a particular card number
11. If the card is verified a Boolean "*true*" is returned to merchant server and the transaction is successful
12. In order to avoid distributed guessing attacks, there is a limit imposed on invalid transactions.

Future improvements

Future work includes integrating AES optimization into the Stripe library by implementing the Anubis S-box. Developing also a unified payment architecture would enhance security and simplify updates.

Conclusion

Online payment systems are a complex interplay of technology, policy, and human factors. While vulnerabilities persist, advances in cryptographic techniques, authentication protocols, and data storage mechanisms have significantly improved their resilience. By adopting a layered security approach, standardizing payment architectures, and leveraging emerging technologies like AI and blockchain, the industry can mitigate risks and build a more secure financial ecosystem. As attackers grow more sophisticated, proactive measures and continuous innovation will remain essential to safeguarding digital transactions.

References

Abdul Jaleel, J., & Assis, A. (2013). Optimization of AES Encryption Algorithm with S-Box. In *International Journal of Engineering Research and Technology* (Vol. 6, Issue 2). http://www.irphouse.com

Bajaj, R. D., & Gokhale, U. M. (2016). AES ALGORITHM FOR ENCRYPTION. *International Journal of Latest Research in Engineering and Technology (IJLRET) Www.Ijlret.Com* ||, *02*, 63–68. www.ijlret.com

Bobba, S. (2020). Enhancing the Security of Online Card Payment System. *International Journal of Advanced Trends in Computer Science and Engineering*, 9(2), 2055–2059. https://doi.org/10.30534/ijatcse/2020/178922020

Easttom, C. (2017). The RSA Algorithm Explored. In *International Journal of Innovative Research in Information Security (IJIRIS)* (Vol. 01). www.ijiris.com