



G L O B A L R A I N

Practices for Secure Software Report

Table of Contents

DOCUMENT REVISION HISTORY	3
CLIENT.....	3
INSTRUCTIONS.....	ERROR! BOOKMARK NOT DEFINED.
DEVELOPER	3
1. ALGORITHM CIPHER	3
2. CERTIFICATE GENERATION	5
3. DEPLOY CIPHER.....	5
4. SECURE COMMUNICATIONS	6
5. SECONDARY TESTING.....	6
6. FUNCTIONAL TESTING	7
7. SUMMARY	7
8. INDUSTRY STANDARD BEST PRACTICES	7

Document Revision History

Version	Date	Author	Comments
1.0	08-17-25	Elizabeth Marticello	Initial Report

Client



Developer

Elizabeth Marticello

1. Algorithm Cipher

Overview

Artemis Financial wants to use the most current and effective software security with its public web interface to protect its client data and financial information. They specifically want to add a file verification step to its web application to ensure secure communication.

I recommend using the cryptographic hash function SHA-256 to ensure data integrity. This cryptographic algorithm can generate a unique fixed-size checksum based on input data. This can be compared before and after transmission or storage to detect errors or tampering. SHA-256 is widely used due to its resistance against collision attacks, where two different inputs produce the same checksum. This algorithm protects user data, like their financial information, against corruption and unauthorized modification (GeeksForGeeks, 2024).

In addition to integrity checks, Artemis Financial also needs to protect the confidentiality of its data. For this I recommend deploying the symmetric encryption algorithm AES-256. AES-256 uses 256-bit keys and processes data in a fixed-size 128-bit blocks, offering a strong balance of performance and security. Unlike hash functions, AES-256 is a cipher that can both encrypt and decrypt data, ensuring only authorized parties can view the information. Its large key size provides resistance against brute force attacks. Together, SHA-256 for verification and AES-256 for encryption create a layered approach to security, addressing both tamper detection and data protection.

Hash Functions and Bit Levels

SHA-256 and AES-256 are both essential to modernizing Artemis Financial's operations but serve very different purposes. SHA-256 is a cryptographic hash function that generates a fixed 256-bit hash output regardless of the input size. It does not use a key, making it a one-way function ideal for data integrity verification (MojoAuth, n.d.-a).

In contrast, AES-256 is a symmetric encryption algorithm that relies on a 256-bit key to encrypt and decrypt data, with its output being ciphertext that matches the size of the original plaintext blocks. While SHA-256 is typically used for checksums, signatures, and ensuring data integrity, AES-256 is designed for confidentiality, securing sensitive data during storage or transmission. Together they provide complementary protection (MojoAuth, n.d.-b).

Random Numbers, Symmetric vs. Non-Symmetric Keys

Random numbers play a critical role in AES encryption by ensuring that generated keys are unpredictable and secure. AES is a symmetric cipher, meaning the same key is used for both encryption and decryption. This approach differs from asymmetric encryption, such as RSA, which uses a public and private key pair. Because symmetric encryption relies on one shared key, strong randomness is essential. If the key is compromised, both the encrypted data and the ability to decrypt it are exposed. By using cryptographically secure random number generators, AES keys can be made unguessable and robust enough to protect Artemis Financial's sensitive client data (Manico & Detlefsen, 2014).

History and Current State of Encryption Algorithms

Some of the earliest encryption methods date back to ancient Sparta, where scrambled letter arrangements protected military communications. During World War I, similar techniques were used to hide military telegram codes. In the 1970s, the Data Encryption Standard (DES) became the first cryptosystem officially used by the U.S. government. However, DES was eventually replaced in 2001 by AES due to its much longer key lengths and stronger resistance to modern hardware attacks (Schneider, n.d.).

Given its proven history, robust key sizes, reliance on strong random numbers, and efficiency in symmetric encryption, AES is the most secure and appropriate cipher for Artemis Financial's long-term file encryption needs.

2. Certificate Generation

Screenshots of generated self-signed certificates using the Java Keytool.

```
Administrator: Command Prompt

Generating 4,096 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 365 days
for: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -export -alias CS305_Project2 -storepass SNHU_CS305_
7.1 -file server.cer -keystore keystore.jks
Certificate stored in file <server.cer>

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -printcert -file server.cer
Owner: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US
Issuer: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US
Serial number: ae7ebc697603bf0e
Valid from: Fri Aug 22 16:23:06 EDT 2025 until: Sat Aug 22 16:23:06 EDT 2026
Certificate fingerprints:
    SHA1: 7C:69:47:D9:0A:6A:6E:9E:98:29:D0:54:33:F8:D2:ED:6A:AE:D1:E1
    SHA256: 21:9F:5D:B8:F0:2B:63:94:A3:59:84:91:BA:D9:1D:0A:74:91:E5:87:6B:71:A5:59:5C:35:3F:6B:6C:96:31:62
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: DA 92 FA 1F B5 81 06 17  0B 3D 40 CA F5 32 79 B5  .....=@..2y.
0010: 92 34 AF DB                .4..
]
]

C:\Windows\system32>
```

```
Administrator: Command Prompt

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -genkeypair -alias CS305_Project2.1 -keyalg RSA -key
size 4096 -sigalg SHA256withRSA -dname "CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, S=NY, C=US" -ext SAN=DNS
:localhost,IP:127.0.0.1 -keystore keystore.jks -storepass SNHU_CS305_7.1 -keypass SNHU_CS305_7.1 -validity 365
Generating 4,096 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 365 days
for: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -export -alias CS305_Project2.1 -file server.cer -ke
ystore keystore.jks -storepass SNHU_CS305_7.1
Certificate stored in file <server.cer>

C:\Windows\system32>
```

3. Deploy Cipher

Screenshot showing the checksum verification.

```
Not secure https://localhost:8443/hash

data:Hello Liz Marticello!

algorithm: SHA-256

checksum: f9105d3f270abdadf3a5416de7d65fbaac899d0aa92371600ccb5804611289f1
```

4. Secure Communications

Screenshot of the web browser showing a secure webpage.



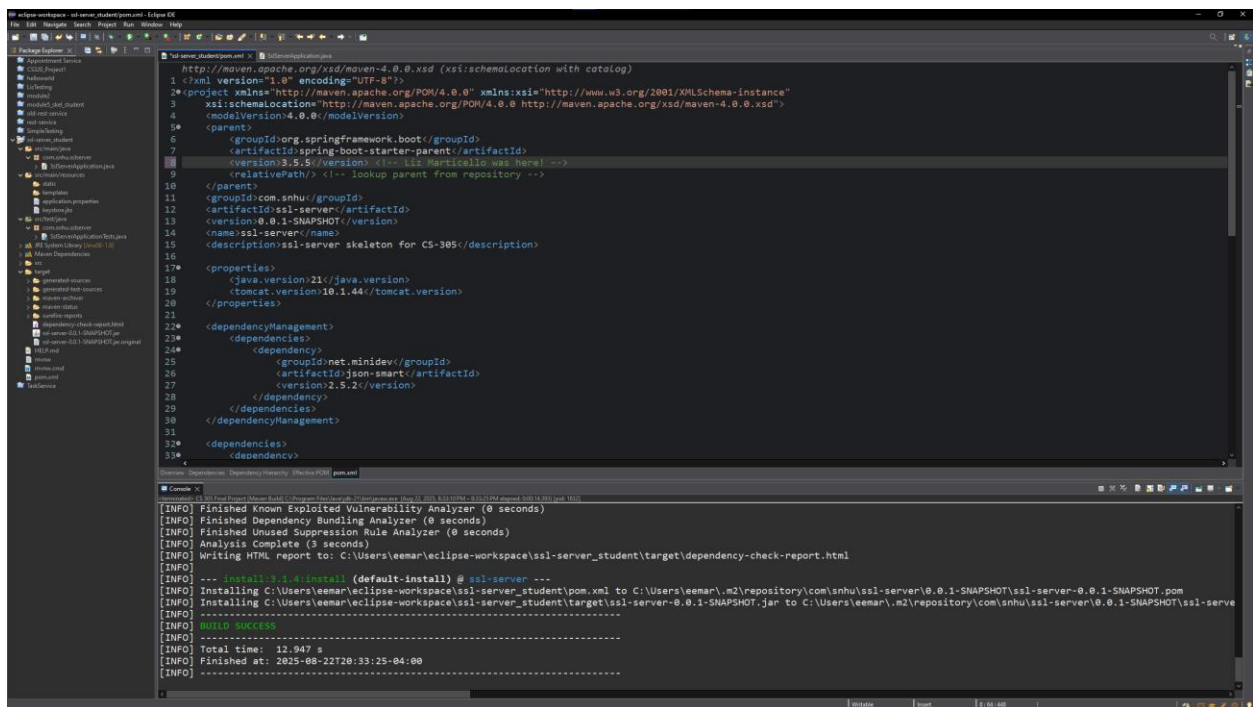
data:Hello Liz Marticello!

algorithm: SHA-256

checksum: f9105d3f270abddadf3a5416de7d65fbaac899d0aa92371600ccb5804611289f1

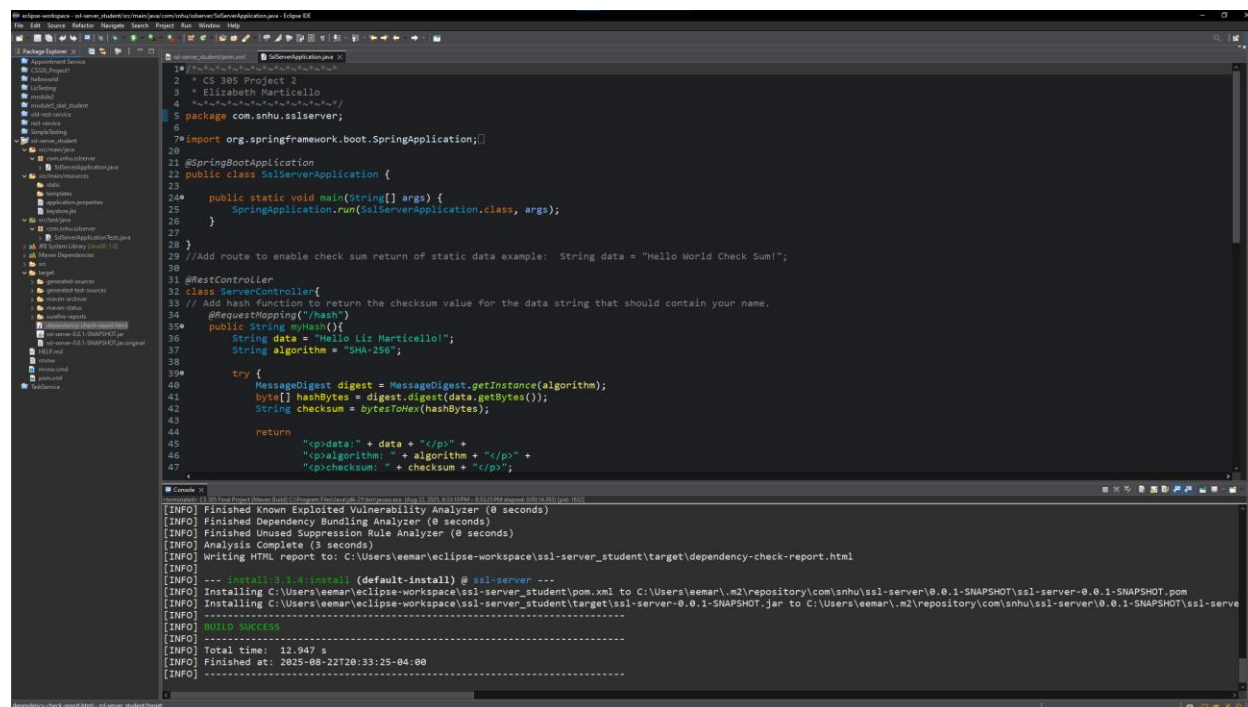
5. Secondary Testing

Screenshots of the refactored code executed without errors and the dependency-check report.



6. Functional Testing

Screenshot showing refactored code executed without errors.



The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows a project named 'ssl-server-student' with various sub-packages like 'com.snhu.sslserver'. The main editor window shows the source code of 'SslServerApplication.java'. The code is a Spring Boot application that implements a REST controller for hashing data. It uses SHA-256 for hashing and includes a main method to run the application. The console at the bottom shows the output of the application, indicating that it ran successfully without any errors. The output includes information about the application's startup, the execution of the REST controller, and the successful completion of the build process.

```
1 // ...  
2 * CS 305 Project 2  
3 * Elizabeth Marticello  
4 * ...  
5 package com.snhu.sslserver;  
  
6  
7 import org.springframework.boot.SpringApplication;  
8  
9  
10  
11 @SpringBootApplication  
12 public class SslServerApplication {  
13  
14     public static void main(String[] args) {  
15         SpringApplication.run(SslServerApplication.class, args);  
16     }  
17  
18 }  
19  
20 //Add route to enable check sum return of static data example: String data = "Hello World Check Sum!";  
21  
22  
23  
24 @RestController  
25 class ServerController {  
26     // Add hash function to return the checksum value for the data string that should contain your name.  
27     @RequestMapping("/hash")  
28     public String myHash() {  
29         String data = "Hello Liz Marticello!";  
30         String algorithm = "SHA-256";  
31  
32         try {  
33             MessageDigest digest = MessageDigest.getInstance(algorithm);  
34             byte[] hashBytes = digest.digest(data.getBytes());  
35             String checksum = bytesToHex(hashBytes);  
36  
37             return "

data: " + data + "</p>" +  
38                 "<p>algorithm: " + algorithm + "</p>" +  
39                 "<p>checksum: " + checksum + "</p>";  
40  
41         } catch (Exception e) {  
42             return "Error: " + e.getMessage();  
43         }  
44     }  
45 }  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000


```

7. Summary

For Artemis Financial's modernization initiative, the refactored codebase was updated to meet current security requirements by focusing on data confidentiality, integrity, and secure communications. Guided by the vulnerability assessment process flow diagram the application was enhanced with SHA-256 hashing to verify file integrity and AES-256 encryption to protect sensitive client and financial data. Secured communications were ensured by implementing HTTPS by generating and deploying a self-signed certificate. Doing this secures all client-server communication and prevents data interception. Input validation and secure API interactions were reviewed to reduce risks of injection or malformed data and error handling was improved to avoid exposing sensitive information. Static testing with dependency-check tools ensured no new vulnerabilities were introduced during refactoring. Functional testing verified that all features, including checksum verification worked as expected. Using both SHA-256 and AES-256

8. Industry Standard Best Practices

Industry Standard best practices, including OWASP guidelines and secure SDLC principles, were carefully applied to preserve the application's existing security while modernizing its overall security. Input validation, parameterized queries, and secure error handling were used to block common attacks like injection. Sensitive data such as encryption keys and credentials were removed from the code and stored securely, and HTTPS was enabled to protect data in transit. Existing access controls, roles, and sessions were kept in place and tested to make sure no new vulnerabilities were added.

These practices make Artemis Financial stronger by lowering the chance of data breaches and building client trust. They also support compliance with common compliance requirements, help avoid costly remediation, and improve the system's reliability and maintainability. Overall, using secure coding practices keeps the platform safe, protecting both company's reputation and its sensitive financial data.

References

(2024, May 28). *Understanding Checksum Algorithm for Data Integrity*. GeeksForGeeks. <https://www.geeksforgeeks.org/system-design/understanding-checksum-algorithm-for-data-integrity/>

MojoAuth (n.d.-a). *SHA-256 vs SHA3-256*. MojoAuth. <https://mojoauth.com/compare-hashing-algorithms/sha-256-vs-sha3-256/>

MojoAuth (n.d.-b). *AES-256 Encryption : Java*. <https://mojoauth.com/encryption-decryption/aes-256-encryption--java/>

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java: Building Secure Web Applications*. McGraw Hill Computing. https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/?sso_link=yes&sso_link_from=SNHU

Schneider, J. (n.d.). *A brief history of cryptography: Sending secret messages throughout time*. <https://www.ibm.com/think/topics/cryptography-history>