

# CS763 Project

## 3

Implementing Crypto

Alessandro Allegranzi

---

## Table of Contents

<b><i>Github Repo</i></b> .....	<b>2</b>
<b><i>Report</i></b> .....	<b>2</b>
Security Feature .....	2
Implementation Details .....	3
Testing .....	6
<b><i>References</i></b> .....	<b>9</b>

## Github Repo

Repository: [https://github.com/1-8192/CS763\\_project](https://github.com/1-8192/CS763_project)

Files I added/changed to implement added security:

[Encryption Utility Class](#)

[String Attribute Converter](#)

[Double Attribute Converter](#)

[JWT 4 Hour Expiration Change](#)

## Report

### Security Feature

The Trackr application already featured JWT tokens for authentication and hashed passwords using bcrypt for persistent storage, so I implemented database encryption.

Database encryption is an important security measure employed to safeguard sensitive information stored in databases, like bank account information or identifying information, by converting it into unreadable code that can only be deciphered with the appropriate cryptographic key [3]. The process involves applying encryption algorithms to the data before it is stored, rendering it effectively indecipherable to unauthorized entities. DB encryption protects sensitive data, such as personal information, financial records, and intellectual property, from unauthorized access, ensuring confidentiality and integrity [3].

DB encryption can mitigate system breaches because in the event of a security breach, encrypted data remains incomprehensible to unauthorized parties, limiting the potential damage and protecting the privacy of individuals. Additionally, database encryption aids in compliance with various data protection regulations and standards, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA), which mandate the protection of sensitive information [3].

Very small change, but, additionally, I shortened the expiration date for the JWT token from the default (which I think is 20 minutes) to 10 minutes, to decrease the potential vulnerabilities in that feature.

## Implementation Details

I implemented database encryption using JPA Attribute converters [1]. The Trackr application was built using an H2 database and JPA ORM, so I decided to leverage the AttributeConverter interface to implement encryption.

Before Implementing the attribute converters, I first built out a utility class to handle the encryption. The class file is viewable in [GH here](#); I'll also include the code below:

```
/**
 * Encryption utility class to encrypt data for database storage.
 */
@Component
public class EncryptionUtility {

    // Bringing in the encryption key and initialization vector from
    // application context.

    @Value("${encrypt_key}")
    private String key;

    @Value("${init_vector}")
    private String vector;

    // Using the AES symmetric algorithm for encryption. Used the standard
    // Cipher transformation example from documentation
    // for AES 128 bit encryption.
    private final String cryptoAlgo = "AES/GCM/NoPadding ";
    private final String algo = "AES";

    /**
     * Encrypt string types.
     * @param value String value we want to encrypt
     * @return encrypted string.
     * @throws RuntimeException
     */
    public String encryptString(String value) throws RuntimeException {
        try {
            IvParameterSpec iv = new
            IvParameterSpec(vector.getBytes(StandardCharsets.UTF_8));
            SecretKeySpec spec = new
            SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), algo);

            Cipher cipher = Cipher.getInstance(cryptoAlgo);
            cipher.init(Cipher.ENCRYPT_MODE, spec, iv);

            byte[] encrypted = cipher.doFinal(value.getBytes());
            return Base64.encodeBase64String(encrypted);
        } catch (NoSuchPaddingException | NoSuchAlgorithmException |
            InvalidKeyException
            | InvalidAlgorithmParameterException |
            IllegalBlockSizeException | BadPaddingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```

    }

    /**
     * Method to decrypt string.
     * @param encrypted encrypted string
     * @return decrypted String
     * @throws RuntimeException
     */
    public String decryptString(String encrypted) throws RuntimeException {
        try {
            IvParameterSpec iv = new
IvParameterSpec(vector.getBytes(StandardCharsets.UTF_8));
            SecretKeySpec spec = new
SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), algo);

            Cipher cipher = Cipher.getInstance(cryptoAlgo);
            cipher.init(Cipher.DECRYPT_MODE, spec, iv);

            byte[] original = cipher.doFinal(Base64.decodeBase64(encrypted));
            return new String(original);
        } catch (NoSuchPaddingException | NoSuchAlgorithmException |
InvalidKeyException
                | InvalidAlgorithmParameterException |
IllegalBlockSizeException | BadPaddingException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Encrypt double types.
     * @param value double to encrypt.
     * @return encrypted string.
     */
    public String encryptDouble(double value) {
        return encryptString(Double.toString(value));
    }

    /**
     * Decrypt double.
     * @param encrypted encrypted string.
     * @return double that was decrypted.
     */
    public double decryptDouble(String encrypted) {
        String decryptedValue = decryptString(encrypted);
        if (decryptedValue != null) {
            return Double.parseDouble(decryptedValue);
        }
        return Double.NaN; // Return NaN for failure or invalid input
    }
}

```

Some notes on the above code:

1. I am injecting the encryption key and initialization vector from application context, so as not to store the plain string in the file. (See pitfalls section below about the key)

2. I decided to use the AES128 symmetric algorithm for encryption because it is a standard option in the Cipher class [2]. It is also a widely used, secure standard for encryption that is stronger than DES or Triple DES [4].
3. I am using the Cipher class [2] to run the actual encryption. It is a Java standard and is the core of the Java Cryptographic Extension (JCE) framework [2].
4. Because the application also stores account balances and other sensitive numeric information as double types, I had to implement double encryption and decryption methods as wrappers around the string methods to convert types and make it work.

I then created the Attribute Converters. Here is a link to the [String converter](#). I also had to create a separate Attribute Converter for the [double type](#). Here is the code for one of the classes:

```
/**
 * Class to encrypt string attributes.
 */
public class EncryptionStringConverter implements
AttributeConverter<String, String> {
    @Autowired
    EncryptionUtility encryptionUtility;

    @Override
    public String convertToDatabaseColumn(String s) {
        return encryptionUtility.encryptString(s);
    }

    @Override
    public String convertToEntityAttribute(String s) {
        return encryptionUtility.decryptString(s);
    }
}
```

This was pretty simple as I just had to override two methods from the AttributeConverter interface to convert attributes to columns and vice versa. Those methods are basically wrappers around calls to the encryption utility class I shared above.

The final step was then adding an annotation to the existing Trackr Entities. Here are 2 examples, one for a string type and one for a double type, from the BankAccount entity.

```
@Column(nullable = false, length = 20)
@NotNull(message = CommonConstants.BLANK_ACCOUNT_TYPE)
@Enumerated(EnumType.STRING)
@Convert(converter = EncryptionStringConverter.class)
private ACCOUNT_TYPE accountType;
```

```
@Column(nullable = false, precision = 2)
@PositiveOrZero(message = CommonConstants.INVALID_BALANCE_VALUE)
@Convert(converter = EncryptionDoubleConverter.class)
private double balance;
```

Through the magic of Spring I just had to include the `@Convert` annotation and specify the converter. The framework handles the rest and applies the encryption to those database columns.

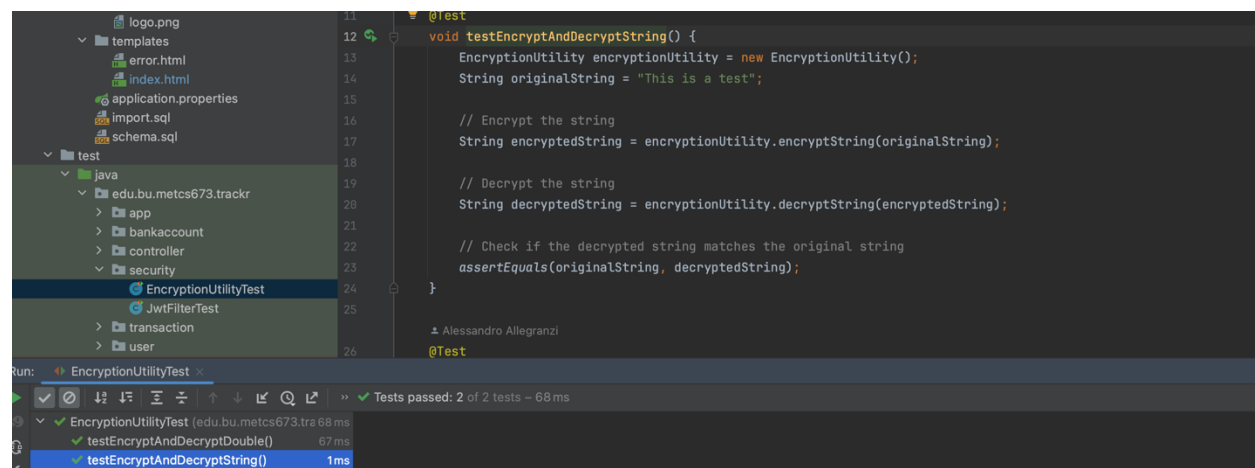
The main pitfall of this implementation is the exposed encryption key. Due to time constraints, I wasn't able to look into storing the key in Hashicorp Vault, or another secret manager. Storing the key in plaintext in the Github repository is a very big issue and would not be a production-ready implementation of the feature. I included it in the [application.properties](#) file to inject into context for now, but ideally I would be loading that from a key manager rather than hard coding the string. Long-term, I would look into storing the key in vault and injecting it into application configs. Another negative aspect is I used the standard Cipher class 128 bit AES algo. Using AES256 would have been a more secure option, and for storing user bank account information, the safest option possible would be best.

I also made a small change to the existing JWT token implementation to set the expiration time to 10 minutes rather than the standard 20. It was a [one line change](#), but easy wins are still wins. The JWT Util class was also injecting an application property strangely, so I cleaned that up by using the `@Value` annotation to reference the variable stored in the application file rather than the string itself. I also removed the redundant constructor.

```
@Value("${jwt_secret}")
public String jwtSecret;
```

## Testing

I added unit tests for the encryption classes for some automated test coverage. Below is a screenshot of the tests passing, showing encryption works. I also tested the application locally manually, and database interaction still worked well.



I also ran a ZAP DAST scan as part of a github action with my code merge. The full scan is available as a downloadable [zip file here](#). The results do not show any sql injection vulnerabilities, which is a good sign. ZAP technically does alert SQL injection and other DB vulnerabilities, but it may not be a great tool for identifying database vulnerabilities mitigated by encryption [5]. Screenshot of the summary below:

#### Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	10
False Positives:	0

#### Alerts

Name	Risk Level	Number of Instances
<a href="#">Content Security Policy (CSP) Header Not Set</a>	Medium	1
<a href="#">Missing Anti-clickjacking Header</a>	Medium	1
<a href="#">Cookie No HttpOnly Flag</a>	Low	1
<a href="#">Cookie without SameSite Attribute</a>	Low	1
<a href="#">Permissions Policy Header Not Set</a>	Low	2
<a href="#">Base64 Disclosure</a>	Informational	1
<a href="#">Information Disclosure - Suspicious Comments</a>	Informational	13
<a href="#">Loosely Scoped Cookie</a>	Informational	2
<a href="#">Modern Web Application</a>	Informational	1
<a href="#">Non-Storable Content</a>	Informational	4
<a href="#">Sec-Fetch-Dest Header is Missing</a>	Informational	3
<a href="#">Sec-Fetch-Mode Header is Missing</a>	Informational	3
<a href="#">Sec-Fetch-Site Header is Missing</a>	Informational	3
<a href="#">Sec-Fetch-User Header is Missing</a>	Informational	3
<a href="#">Session Management Response Identified</a>	Informational	3

I ran a Sonarqube SAST scan locally, which showed a vulnerability related to the original encryption algorithm I was using.



**sonarqube** Projects Issues Rules Quality Profiles Quality Gates Administration More Q

☆ test / main ✓ ? ⚠ The last analysis has warnings. [See details](#) Version not provided 🔗

Overview **Issues** Security Hotspots Measures Code Activity Project Settings ▾ Project Information

My Issues All

**Filters** Clear All Filters

Issues in new code

▼ **Clean Code Attribute**

Consistency	0
Intentionality	0
Adaptability	0
Responsibility	2

☐ Bulk Change Select issues ▾ ▾ Navigate to issue 1 2 **2 issues** **40min effort**

src/.../bu/metcs673/trackr/security/EncryptionUtility.java

Responsibility issue

☐ [Use another cipher mode or disable padding.](#) cwe privacy +

Open ▾ Not assigned ▾ Security 🔴 Vulnerability Critical 20min effort · 1 day ago

Responsibility issue

☐ [Use another cipher mode or disable padding.](#) cwe privacy +

Open ▾ Not assigned ▾ Security 🔴 Vulnerability Critical 20min effort · 1 day ago

2 of 2 shown

**Use another cipher mode or disable padding.** 🔗

Encryption algorithms should be used with secure mode and padding scheme [java:S5542](#)

Software qualities impacted: Security 🔴

Open ▾ Not assigned ▾ Vulnerability Critical

cwe ... +

**Effort**  
20min

**Introduced**  
1 day ago

**Where is the issue?** **Why is this an issue?** **How can I fix it?** **Activity** **More Info**

```

48         cipher.init(Cipher.ENCRYPT_MODE, spec, iv);
49
50         byte[] encrypted = cipher.doFinal(value.getBytes());
51         return Base64.encodeBase64String(encrypted);
52     } catch (NoSuchPaddingException | NoSuchAlgorithmException | InvalidKeyException
53             | InvalidAlgorithmParameterException | IllegalBlockSizeException |
54             BadPaddingException e) {
55         throw new RuntimeException(e);
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   ...

```

I followed Sonarqube's suggestion and swapped out the "AES/CBC/PKCS5PADDING" algorithm for "AES/GCM/NoPadding" crypto algorithm for Cipher.


The successive scan showed the vulnerabilities were resolved.

☆ test / ⓘ main ✓ ?


The last analysis has warnings. [See details](#) Version not provided

Overview Issues Security Hotspots Measures Code Activity Project Settings Project Information

### Quality Gate Status ?



Quality Gate  
**Passed**



Enjoy your sparkling clean code!

### Measures

New Code Overall Code

New Code: Since December 4, 2023 Started 3 minutes ago

<b>Reliability</b> 0 New Bugs <span>A</span>	<b>Maintainability</b> 0 New Code Smells <span>A</span>
<b>Security</b> 0 New Vulnerabilities <span>A</span>	<b>Security Review</b> 0 New Security Hotspots ? <span>A</span>

## References

[1] “JPA Attribute Converters” *Baeldung*. <https://www.baeldung.com/jpa-attribute-converters>

[2] “Class Cipher” *Oracle Docs*.  
<https://docs.oracle.com/javase/8/docs/api/javax/crypto/Cipher.html>

[3] Gulen, Kerem. “Cracking the code: How database encryption keeps your data safe?” *Dataconomy*. <https://dataconomy.com/2023/04/11/what-is-database-encryption-types-methods/>  
 April 11, 2023.

[4] “Advanced Encryption Standard (AES)” *GeeksforGeeks*.  
<https://www.geeksforgeeks.org/advanced-encryption-standard-aes/>

[5] “SQL Injection” *ZAP Docs*. <https://www.zaproxy.org/docs/alerts/40018/>