Ingegneria del Software 2

Repositories

BookKeeper:*https://github.com/Ludovix9070/bookkeeper*

Syncope:*https://github.com/Ludovix9070/syncope*

Report Software Testing

0320460

Ludovico De Santis

Indice

1 BookKeeper

WRITE CACHE

La classe WriteCache ha l'obiettivo di migliorare le prestazioni della scrittura dei dati su disco. Infatti, prima di effettuare la scrittura su disco, il dato viene memorizzato nella cache. La memoria viene opportunamente suddivisa in segmenti multipli, al fine di migliorarne l'accesso e l'indicizzazione. Le entries vengono aggiunte in un buffer comune e indicizzate attraverso un Hashmap finchè la cache viene pulita.

public boolean put(long ledgerId, long entryId, ByteBuf entry)

Lo scopo di questo metodo è quello di inserire un dato nella cache di scrittura, copiando nella cache il contenuto del ByteBuf entry, identificato da entryId e da scrivere nel Legder specificato da ledgerId. Il metodo ritorna come valore true in caso di avvenuta scrittura oppure false in caso contrario. L'output atteso è stato ottenuto tramite un'analisi white-box,a causa della mancanza di documentazione. Per quanto riguarda la entry ByteBuf, essendo un parametro complesso, è stato effettuato un approccio di suddivisione di entry nulla, valida e invalida. La suddivisione delle classi di equivalenza è stata effettuata come illustrato in tabella.

Classi di equivalenza

Parametro	Classi
ledgerId	{ledgerId <0};{ledgerId ≥ 0};
entryId	{entryId <0};{entryId ≥ 0};
entry	{null};{valid};{invalid};

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	Boundaries
ledgerId	{-1};{0};{1}
entryId	{-1};{0};{1}
entry	{null};{valid};{invalid};

Come entry invalida, è stata ideata una entry di dimensioni maggiori alla dimensione massima della cache. L' utilizzo di un approccio multidimensionale è stato valutato come ideale per coprire tutte le possibili situazioni ed avere una copertura delle casistiche ottimale.

ledgerId	entryId	entry	output
-1	-1	Null	NullPointerException
-1	-1	Valid	IllegalArgumentException
-1	-1	Invalid	False
-1	0	Null	NullPointerException
-1	0	Valid	IllegalArgumentException
-1	0	Invalid	False
-1	1	Null	NullPointerException

-1	1	Valid	IllegalArgumentException
-1	1	Invalid	False
0	-1	Null	NullPointerException
0	-1	Valid	IllegalArgumentException
0	-1	Invalid	False
0	0	Null	NullPointerException
0	0	Valid	True
0	0	Invalid	False
0	1	Null	NullPointerException
0	1	Valid	True
0	1	Invalid	False
1	-1	Null	NullPointerException
1	-1	Valid	IllegalArgumentException
1	-1	Invalid	False
1	0	Null	NullPointerException
1	0	Valid	True
1	0	Invalid	False
1	1	Null	NullPointerException
1	1	Valid	True
1	1	Invalid	False

Analisi dei Test

Dopo l'esecuzione dei test con le configurazioni precedentemente illustrate, basate su analisi white-box, si è effettuata un'analisi della coverage tramite JaCoCo, ottenendo una *statement coverage* del 96%, con una *branch coverage* del 62% (Figura1). Al fine di aumentare la coverage, soprattutto la branch coverage, è stata utilizzata una istanza di cache non più con dimensione massima dei segmenti impostata a valori di default, ma con dei valori custom. In questo modo, si è ottenuta una *statement coverage* del 98% ed una *branch coverage* del 87% (Figura2), migliorando notevolmente le percentuali precedenti. In particolare, si può osservare un analisi di dettaglio in Figura3. Non è stato possibile arrivare ad una coverage del 100% in quanto necessario uno scenario di scrittura in contemporanea che mandasse in stato di errore una delle due parti sulla "compareAndSet". Si è passati quindi ad un'analisi di tipo *data flow coverage,* ottenendo come risultato che si raggiungono come coverage 51 su 56 def-use, come si può notare dalla Figura4. Successivamente, è stata introdotta un'analisi di tipo *mutation coverage,* il cui risultato è visibile in Figura5 e Figura6. Quindi, è stato progettato un caso di test per eliminare il mutante a riga 150 (changed conditional boundary), utilizzando una size di dimensione uguale alla cache capability. Inoltre, è stato introdotto un controllo sul numero di elementi in cache per eliminare il mutante a riga 181. I risultati finali portano quindi ad un miglioramento della nostra test suite, come si può evincere dalla Figura7 e Figura8.

public ByteBuf get(long ledgerId, long entryId)

Questo metodo prende in input l'id del ledger e l'id della entry, ritornando il contenuto della entry se questa è presente in cache. L'output atteso è stato ottenuto tramite un'analisi white-box, a causa della mancanza di documentazione.

Classi di equivalenza

Parametro	Classi
ledgerId	{ledgerId <0};{ledgerId ≥ 0};
entryId	{entryId <0};{entryId ≥ 0};

Boundary Analysis

I valori boundary selezionati per i parametri sono i seguenti:

Parametro	Boundaries
ledgerId	{-1};{0};{1}
entryId	{-1};{0};{1}

L' utilizzo di un approccio multidimensionale è stato valutato come ideale per coprire tutte le possibili situazioni ed avere una copertura delle casistiche ottimale.

ledgerId	entryId	output
-1	-1	IllegalArgumentException
-1	0	IllegalArgumentException
-1	1	IllegalArgumentException
0	-1	Null
0	0	ValidEntry
0	1	ValidEntry
1	-1	Null
1	0	ValidEntry
1	1	ValidEntry

Analisi dei test

Dopo l'esecuzione dei test con le configurazioni precedentemente illustrate, basate su analisi white-box, si è proceduto ad effettuare un'analisi della coverage con JaCoCo. Si è ottenuto quindi una *statement coverage* ed una *branch coverage* del 100%(Figura1, dettaglio Figura9), segno dell'ottima progettazione della test suite. Si è passati quindi ad un'analisi di tipo *data flow coverage*, con coperte tutte le def-use, come si può notare in Figura10. In un'analisi di tipo *mutation coverage,* tutti i mutanti risultano eliminati (Figura11).

DIGEST MANAGER

Bookkeeper fornisce il supporto per l'utilizzo di diversi tipi di algoritmi per la generazione e la gestione di Digest, sia a 32 bit (e.g. CRC32, CRC32C) che a 64 bit (e.g. HMAC), finalizzate al controllo di integrità per le entry che vengono memorizzate.

La classe Digest Manager si occupa, in fase di invio, di calcolare il digest da allegare ai pacchetti da inviare e, in ricezione, di verificare la correttezza del pacchetto ricevuto.

private ByteBufList computeDigestAndPackageForSendingV3(long entryId, long lastAddConfirmed, long length, ByteBuf data)

L' obiettivo del metodo è quello di calcolare il digest di una entry e concatenarlo con il buffer di dati associato all'entry stessa. Tramite domain partitioning è stata valutata una prima partizione in classi di equivalenza, basata sulla semantica del nome dei parametri in input e sulle tipologie di dato.

Classi di equivalenza

Parametro	**Classi**
entryId	{entryId ≤ lastAddConfimed}; {entryId > lastAddConfirmed};
lastAddConfirmed	{lastAddConfirmed < 0};{lastAddConfirmed ≥ 0};
length	{length ≤ 0}; {length > 0};
data	{null}; {valid}; {invalid};

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	**Boundaries**
entryId	{lastAddConfirmed-1};{lastAddConfirmed};{lastAddConfirmed+1}
lastAddConfirmed	{-1};{0};{1}
length	{-1};{0};{1}
data	{null}; {valid}; {invalid};

Un approccio di tipo monodimensionale è stato utilizzato per lo sviluppo dei casi di test.

entryId	lastAddConfirmed	length	data	output
-1	0	-1	null	NullPointerException
1	1	0	ValidData	BytebBufListValid
0	-1	1	InvalidData	IndexOutOfBoundException

Analisi dei test

A causa della mancanza di documentazione, i valori attesi in caso di dati invalidi o di data impostata a null sono stati ricavati utilizzando un approccio white-box. Da una prima analisi (Figura12, dettaglio in Figura13), risulta una statement coverage del 65%. A seguito dell'analisi della coverage, sono stati quindi progettati I seguenti casi di test:

entryId	lastAddConfirmed	length	data	output
0	1	0	CompositeByteBufData	ByteBufListValid
0	1	0	WrappedData	ByteBufListValid

Grazie a questi miglioramenti, si è riusciti ad ottenere una *statement coverage* del 96% ed una *branch coverage* del 87% (Figura14, dettaglio Figura15), aumentando notevolmente le percentuali precedenti. Si è passati quindi ad un'analisi di tipo *data flow coverage*, con coverage di 30 def-use coperte su 33 (Figura16). Successivamente, si è passati ad un'analisi sulla *mutation coverage*, come mostrato in Figura17. Non si è riuscito ad eliminare 3 mutanti, in particolare nel primo caso (riga 172), dove viene sostituita l'addizione con la sottrazione, la scrittura non va in IndexoutOfBoundException, nonostante avvenga una scrittura di dati con dimensione superiore alla memoria allocata in fase di creazione del buffer.

private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)

Questo metodo permette al server, in ricezione, di verificare il digest del pacchetto ricevuto dal client ed ottenere l'entry corrispondente, rimuovendo l'header dovuto al digest. Il metodo, essendo privato, è stato testato utilizzando il metodo pubblico *public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)*, che dopo aver verificato la correttezza del digest ritorna il buffer di dati associato. Tramite un'analisi whitebox, si nota che il metodo chiama il metodo privato con skipEntryIdCheck settato a false. Per questo motivo, il parametro non è stato preso in considerazione. Le classi di equivalenza individuate sono quindi le seguenti:

Classi di equivalenza

Parametro	Classi
entryId	{entryId <0};{entryId ≥ 0};
dataReceived	{null};{valid};{invalid};

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	Boundaries
entryId	{-1};{0};{1}
dataReceived	{null};{valid};{invalid};

L' utilizzo di un approccio multidimensionale è stato valutato come ideale per coprire tutte le possibili situazioni ed avere una copertura delle casistiche ottimale.

entryId	dataReceived	output
-1	Null	NullPointerException
-1	Invalid	BKDigestMatchException
-1	Valid	BKDigestMatchException
0	Null	NullPointerException
0	Invalid	BKDigestMatchException
0	Valid	ValidData
1	Null	NullPointerException
1	Invalid	BKDigestMatchException
1	Valid	ValidData

Nonostante il metodo *verifyDigest* torni void, è stato stabilito come valore atteso,in caso di successo, un buffer di dati validi, in quanto è il valore di ritorno della funzione pubblica utilizzata per testare il metodo privato.

Un parametro dataReceived invalido è stato ottenuto creando un buffer di dati con all'interno un entryId diverso da quello specificato come primo parametro in input. Come algoritmo per il calcolo del digest è stato scelto Hmac, tale algoritmo viene scelto al momento dell'istanziazione della classe Digest Manager.

Analisi dei test

Dopo l'esecuzione dei test con le configurazioni precedentemente illustrate, tramite JaCoCo otteniamo sul metodo una coverage come illustrato in Figura18. Per migliorare la coverage, sono stati progettati nuovi casi di test. In particolare, per il branch a riga 246, è stata creata una entry minore della lunghezza minima del buffer di dati, pari alla dimensione del buffer per i metadati più la dimensione del digest (nel caso di utilizzo di Hmac pari a 20 bytes). Per il branch a riga 258, è stata utilizzata una ulteriore istanza della classe DigestManager, basata questa volta su un algoritmo a 32 bit, il CRC32C, per testare il corretto funzionamento del metodo anche con digest basati su algoritmi a 32 bit. In entrambi i casi (32 e 64 bit), per creare istanze errate rispettivamente con errori di tipo "Digest mismatch" (32 bit, riga 260) e di tipo "Mac mismatch" (64 bit, riga 269), e conseguentemente quindi aumentare la coverage, sono stati creati dei digest volutamente errati basati su algoritmi differenti rispetti a quelli dell'istanza in uso, ottenendo quindi l'effetto voluto. Inoltre, per il branch a riga 278, è stato creato un buffer con ledgerId diverso da quello dell'istanza della classe digest manager in uso. Infine, risultava non coperto un branch a riga 284, questo perché il metodo pubblico utilizzato per testare il metodo privato, imposta sempre il valore di skipEntryIdCheck a false. Per ovviare a questo e migliorare ulteriormente la coverage, si è progettato un test in cui era previsto il settaggio del valore di skipIdCheck a true. Si progetta quindi un caso di test valido utilizzando quindi il metodo pubblico verifyDigestAndReturnLastConfirmed(ByteBuf dataReceived), che ritorna un tipo di dato *Recovery Data* , ovvero un buffer di dati. I caso di test progettati sono riassunti nella tabella sottostante:

entryId	dataReceived	output
1	ShortEntry	BKDigestMatchException
1	InvalidMacEntry	BKDigestMatchException
1	InvalidDigestEntry	BKDigestMatchException
1	InvalidLedgerIdEntry	BKDigestMatchException
1	Valid32BitEntry	ValidData
1	ValidSkipTrueEntry	ValidData

Dopo aver implementato questi casi di test, si raggiunge una statement coverage ed una branch coverage del 100%, sia per il metodo verifyDigest sia per i metodi pubblici utilizzati per testarlo, come visibile nelle figure 19, 20 e 21. Successivamente, si è passati ad un'analisi di tipo *data-flow coverage,* ottenendo un ottimo risultato con tutte le coppie def-use coperte, come visibile nella Figura22 e nella Figura23. Inoltre, è stata effettuata una analisi di tipo *mutation coverage*. Per un mutante a riga 246 di tipo "changed conditional boundary", è stato progettato un caso di test dove viene passato un buffer di dimensioni identiche a quelle di MetadataLength+macCodeLength (32+20 in caso di Hmac). A seguito di questo miglioramento, il metodo risulta con tutti i mutanti eliminati (Figura24).

2 Syncope

Apache Syncope è un sistema open-source ideato per la gestione delle identità digitali, con l'obiettivo di gestire i dati e le informazioni degli utenti e dei loro account su sistemi e applicazioni. Permette, inoltre, di definire politiche di accesso alle risorse a livello di singolo utente, gruppi di utenti o ad altri oggetti che si possono definire in maniera personalizzata (es. stampanti).

ENCRYPTOR

La prima classe che si è deciso di testare è **Encryptor,** che ha come scopo quello di fornire supporto per l'hashing o cifratura delle password, in base a differenti algoritmi selezionabili. La classe non è documentata in maniera precisa, è stato necessario reperire informazioni sparse all'interno della documentazione generale del progetto, andando ad integrare queste informazioni ove necessario con il codice sorgente, adottando una strategia grey-box. In particolare, la documentazione è molto dettagliata sugli algoritmi utilizzati:

- SHA
- SHA1
- SHA256
- SHA512
- AES
- SMD5
- SSHA
- SSHA1
- SSHA256
- SSHA512
- BCRYPT

I metodi selezionati per i test sono:

- public String **encode** (final String vaue, final CipherAlgorithm cipherAlgorithm)
- public boolean **verify** (final String value, final CipherAlgorithm cipherAlgorithm, final string encoded)
- public String **decode** (final String encoded, final CipherAlgorithm cipherAlgorithm)

i metodi e i relativi casi di test verranno analizzati in dettaglio nelle pagine successive.

public String encode(finalString value, final CipherAlgorithm cipherAlgorithm)

Il primo metodo scelto è il metodo “encode”, che ha il compito di effettuare la cifratura. Infatti il primo parametro in input è la stringa da cifrare, mentre il secondo rappresenta l’algoritmo da utilizzare per effettuare la cifratura stessa. La documentazione inoltre fa riferimento all’utilizzo di una “secret key” necessaria per la sola cifratura utilizzando AES, che infatti è un algoritmo a chiave simmetrica, ma per i nostri casi di test verrà usata una secret key impostata di default dal sistema.

Classi di equivalenza

Parametro	**Classi**
value	{null};{empty};{not empty}
cipherAlgorithm	{null};{SHA};{SHA1};{SHA256};{SHA512};{AES};{SMD5};{SSHA};{SSHA1};{SSHA256};{SSHA512};{BCRYPT};

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	**Boundaries**
value	{null};{“”};{randomString}
cipherAlgorithm	{null};{SHA};{SHA1};{SHA256};{SHA512};{AES};{SMD5};{SSHA};{SSHA1};{SSHA256};{SSHA512};{BCRYPT};

L’ utilizzo di un approccio multidimensionale è stato valutato come ideale per coprire tutte le possibili situazioni ed avere una copertura delle casistiche ottimale.

value	**cipherAlgorithm**	**output**
Null	Null	Null
Null	SHA	Null
Null	SHA1	Null
Null	SHA256	Null
Null	SHA512	Null
Null	AES	Null
Null	SMD5	Null
Null	SSHA	Null
Null	SSHA1	Null
Null	SSHA256	Null
Null	SSHA512	Null

Null	BCRYPT	Null
“”	Null	ValidString
“”	SHA	ValidString
“”	SHA1	ValidString
“”	SHA256	ValidString
“”	SHA512	ValidString
“”	AES	ValidString
“”	SMD5	ValidString
“”	SSHA	ValidString
“”	SSHA1	ValidString
“”	SSHA256	ValidString
“”	SSHA512	ValidString
“”	BCRYPT	ValidString
randomString	Null	ValidString
randomString	SHA	ValidString
randomString	SHA1	ValidString
randomString	SHA256	ValidString
randomString	SHA512	ValidString
randomString	AES	ValidString
randomString	SMD5	ValidString
randomString	SSHA	ValidString
randomString	SSHA1	ValidString
randomString	SSHA256	ValidString
randomString	SSHA512	ValidString
randomString	BCRYPT	ValidString

Come stringa non vuota è stata utilizzata una stringa generata randomicamente. Il valore di ritorno è rappresentato dalla stringa risultato della codifica effettuata dal metodo sulla stringa passata in input secondo l’algoritmo specificato.

Analisi dei test

Dopo l’esecuzione dei test (effettuati utilizzando anche Mockito), si nota che il metodo, in caso di stringa “null”, torna null, come si può evincere da un’analisi del codice. Inoltre, in caso di CipherAlgorithm impostato a null, si effettuala codifica di default con AES. Effettuando una prima analisi con JaCoCo (Figura25, dettaglio Figura26), si ottiene una coverage del 100%, segno dell’ottimo comportamento della TestSuite. Successivamente, si è passati ad un’analisi basata su *mutation coverage,* ottenendo ottimi risultati. Infatti, tutti i mutanti risultano eliminati, come si può evincere dal dettaglio in Figura27.

public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded)

Lo scopo del metodo verify è quello di verificare che l'encoding della stringa value effettuato precedentemente e passato come parametro "encoded" sia stato effettuato correttamente, rispettando l'algoritmo di cifratura indicato.

Classi di equivalenza

Parametro	**Classi**
value	{null};{empty};{not empty}
cipherAlgorithm	{null};{SHA};{SHA1};{SHA256};{SHA512};{AES};{SMD5}; {SSHA};{SSHA1};{SSHA256};{SSHA512};{BCRYPT};
encoded	{correctlyEncoded};{badEncoded}

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	**Boundaries**
value	{null};{""};{randomString}
cipherAlgorithm	{null};{SHA};{SHA1};{SHA256};{SHA512};{AES}; {SMD5};{SSHA};{SSHA1};{SSHA256};{SSHA512}; {BCRYPT};
encoded	{CorrectEncodedValue};{BadEncoded};

Come stringa non vuota è stata utilizzata una stringa randomica, come CorrectEncodedValue è stato utilizzato un corretto encoding di value utilizzando l'algoritmo di cifratura specificato come secondo parametro (CorrectEncodedValue = encode(value,cipherAlgorithm), mentre come BadEncoded è stata utilizzata una stringa randomica, diversa da quella specificata in value.

L' utilizzo di un approccio multidimensionale è stato valutato come ideale per coprire tutte le possibili situazioni ed avere una copertura delle casistiche ottimale.

value	**cipherAlgorithm**	**encoded**	**output**
Null	Null	CorrectEncodedValue	False
Null	SHA	CorrectEncodedValue	False
Null	SHA1	CorrectEncodedValue	False
Null	SHA256	CorrectEncodedValue	False

Null	SHA512	CorrectEncodedValue	False
Null	AES	CorrectEncodedValue	False
Null	SMD5	CorrectEncodedValue	False
Null	SSHA	CorrectEncodedValue	False
Null	SSHA1	CorrectEncodedValue	False
Null	SSHA256	CorrectEncodedValue	False
Null	SSHA512	CorrectEncodedValue	False
Null	BCRYPT	CorrectEncodedValue	False
Null	Null	BadEncoded	False
Null	SHA	BadEncoded	False
Null	SHA1	BadEncoded	False
Null	SHA256	BadEncoded	False
Null	SHA512	BadEncoded	False
Null	AES	BadEncoded	False
Null	SMD5	BadEncoded	False
Null	SSHA	BadEncoded	False
Null	SSHA1	BadEncoded	False
Null	SSHA256	BadEncoded	False
Null	SSHA512	BadEncoded	False
Null	BCRYPT	BadEncoded	False
“”	Null	CorrectEncodedValue	True
“”	SHA	CorrectEncodedValue	True
“”	SHA1	CorrectEncodedValue	True
“”	SHA256	CorrectEncodedValue	True
“”	SHA512	CorrectEncodedValue	True
“”	AES	CorrectEncodedValue	True
“”	SMD5	CorrectEncodedValue	True
“”	SSHA	CorrectEncodedValue	True
“”	SSHA1	CorrectEncodedValue	True
“”	SSHA256	CorrectEncodedValue	True
“”	SSHA512	CorrectEncodedValue	True
“”	BCRYPT	CorrectEncodedValue	True
“”	Null	BadEncoded	False
“”	SHA	BadEncoded	False
“”	SHA1	BadEncoded	False
“”	SHA256	BadEncoded	False
“”	SHA512	BadEncoded	False
“”	AES	BadEncoded	False
“”	SMD5	BadEncoded	False

“”	SSHA	BadEncoded	False
“”	SSHA1	BadEncoded	False
“”	SSHA256	BadEncoded	False
“”	SSHA512	BadEncoded	False
“”	BCRYPT	BadEncoded	False
randomString	Null	CorrectEncodedValue	True
randomString	SHA	CorrectEncodedValue	True
randomString	SHA1	CorrectEncodedValue	True
randomString	SHA256	CorrectEncodedValue	True
randomString	SHA512	CorrectEncodedValue	True
randomString	AES	CorrectEncodedValue	True
randomString	SMD5	CorrectEncodedValue	True
randomString	SSHA	CorrectEncodedValue	True
randomString	SSHA1	CorrectEncodedValue	True
randomString	SSHA256	CorrectEncodedValue	True
randomString	SSHA512	CorrectEncodedValue	True
randomString	BCRYPT	CorrectEncodedValue	True
randomString	Null	BadEncoded	False
randomString	SHA	BadEncoded	False
randomString	SHA1	BadEncoded	False
randomString	SHA256	BadEncoded	False
randomString	SHA512	BadEncoded	False
randomString	AES	BadEncoded	False
randomString	SMD5	BadEncoded	False
randomString	SSHA	BadEncoded	False
randomString	SSHA1	BadEncoded	False
randomString	SSHA256	BadEncoded	False
randomString	SSHA512	BadEncoded	False
randomString	BCRYPT	BadEncoded	False

Analisi dei test

Dopo l’esecuzione dei test (effettuati utilizzando anche Mockito), si nota che il metodo, in caso di stringa value impostata a null, torna come valore “false”. Inoltre, anche in questo caso, con CipherAlgorithm a null, si utilizza di default AES come algoritmo. Dall’analisi effettuata con JaCoCo, si evidenzia che si raggiunge una *statement* e *branch coverage* del 100% (Figura28, dettaglio Figura29), a testimonianza dell’ottima base della test suite. In seguito, si è effettuata un’analisi basata su *mutation coverage* tramite Pit, in cui si può notare come tutti i mutanti risultino eliminati (Figura30).

public String decode(final String encoded, final CipherAlgorithm cipherAlgorithm)

il metodo in questione, come indicato in documentazione, si può utilizzare con successo solo con stringhe cifrate con AES, che infatti è l'unico algoritmo a chiave simmetrica. Sono state individuate quindi le seguenti classi di equivalenza:

Classi di equivalenza

Parametro	**Classi**
encoded	{null};{empty};{wellEncoded};{badEncoded}
cipherAlgorithm	{null};{AES};{notAES};

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	**Boundaries**
encoded	{null};{“”};{wellEncodedString};{badEncodedString}
cipherAlgorithm	{null};{AES};{SHA};

Come stringa ben codificata è stata utilizzata una stringa randomica (originalString) correttamente codificata con AES (wellEncodedString = encode(originalString,AES)), mentre come badEncodedString è stata utilizzata una stringa randomica, diversa da quella utilizzata per creare il corretto encoding precedente. Come Cipher Algorithm diverso da AES è stato invece utilizzato SHA.

L' utilizzo di un approccio multidimensionale è stato valutato come ideale per coprire tutte le possibili situazioni ed avere una copertura delle casistiche ottimale.

encoded	**cipherAlgorithm**	**output**
Null	Null	Null
Null	AES	Null
Null	SHA	Null
“”	Null	Null
“”	AES	“”
“”	SHA	Null
wellEncodedString	Null	Null

wellEncodedString	AES	OriginalString
wellEncodedString	SHA	Null
badEncodedString	Null	Null
badEncodedString	AES	illegalBlockSizeException
badEncodedString	SHA	Null

Analisi dei test

Dopo l'esecuzione dei test (effettuati utilizzando anche Mockito), si nota che, come preannunciato, l'unico caso in cui avviene effettivamente la decodifica si verifica quando la stringa in input è correttamente codificata tramite AES, così come viene correttamente indicato tale algoritmo come secondo parametro. Dall'analisi effettuata con JaCoCo, si evidenzia che si raggiunge una *statement* e *branch coverage* del 100% (Figura31, dettaglio Figura32), a testimonianza dell'ottima base della test suite. In seguito, si è effettuata un'analisi basata su *mutation coverage* tramite Pit, in cui si può notare come tutti i mutanti risultino eliminati (Figura33).

Infine, valutando la coverage di classe (Figura34, dettaglio Figura35), si è deciso di utilizzare una secretkey non di default per aumentare la coverage. Come riportato dalla documentazione, se si specifica una secretKey diversa da quella di default, deve avere lunghezza minima di 16 caratteri, altrimenti viene automaticamente "paddata" dal sistema. Viene quindi deciso di testare questa funzionalità, fornendo in fase di istanziazione della classe Encryptor una secretkey minore di 16 caratteri. La funzionalità risulta corretta, ottenendo inoltre valori di coverage ottimali come indicato in Figura36.

DEFAULTPASSWORDGENERATOR

La seconda classe che si è deciso di testare è **DefaultPasswordGenerator,** che ha come scopo la gestione e la generazione delle password. Tale classe è stata selezionata per la sua importanza, ritenuta fondamentale, nel campo della gestione delle identità digitali. Questa classe ha il compito di generare automaticamente delle password, seguendo determinate policies selezionabili.

public String generate(final List<PasswordPolicy> policies)

Questo metodo ha il compito di generare una password in accordo alla lista di policies passata in input, la password generata viene tornata come parametro di output.

Classi di equivalenza

Parametro	Classi
policies	{null};{empty};{valid};{invalid}

Boundary Analysis

I valori boundary scelti per i parametri sono i seguenti:

Parametro	Boundaries
policies	{null};{emptypoliciesList};{listWithValidPolicy}; {listWithInvalidPolicy}

Essendo il parametro unico, è stato attuato un approccio monodimensionale.

policies	output
null	NullPointerException
emptyPoliciesList	PasswordConformePoliciesDefault
listWithValidPolicy	PasswordConformePolicyIndicate
listWithInvalidPolicy	IllegalArgumentException

Analisi dei test

Dopo l'esecuzione dei test (effettuati utilizzando anche Mockito), si nota che la coverage risulta essere già ottimale per il metodo in esame (Figura37). Si decide allora di aumentare la coverage a livello di classe, andando a soddisfare tutte le condizioni riguardanti le policies presenti nel metodo merge (situazione di dettaglio precedente al miglioramento della test suite consultabili nella Figura38 e nella Figura39).

policies	output
ListPoliciesMinLength	PasswordConformePolicyIndicate
ListPoliciesMaxLength	PasswordConformePolicyIndicate
ListPoliciesUppercase	PasswordConformePolicyIndicate
ListPoliciesLowercase	PasswordConformePolicyIndicate
ListPoliciesDigit	PasswordConformePolicyIndicate
ListPoliciesSpecial	PasswordConformePolicyIndicate
ListPoliciesSpecialChars	PasswordConformePolicyIndicate
ListPoliciesIllegalChars	PasswordConformePolicyIndicate
ListPoliciesRepeatSame	PasswordConformePolicyIndicate
ListPoliciesWordNotPermitted	PasswordConformePolicyIndicate
ListPoliciesMinMax	PasswordConformePolicyIndicate
ListPoliciesNotUppercase	PasswordConformePolicyIndicate

Dopo aver implementato i casi di test con le opportune policies per avere una coverage ottimale, si riesce ad avere una coverage del 100% anche sul metodo Merge (dettaglio Figura40, Figura41).

Per ottenere, inoltre, una coverage di classe ottimale, bisogna utilizzare il metodo *public String generate(final External Resource resource).* Il dettaglio della coverage precedente ai nuovi metodi è consutabile in Figura 42). Sono stati quindi progettati altri due casi di test:

policies	output
ExternalResourceValidPolicies	PasswordConformePolicies
Null	NullPointerException

Dopo l'implementazione di questi casi di test, il metodo risulta coperto al 100% (Dettaglio Figura43), con una ottima coverage finale a livello di classe (Figura44).

Successivamente, si è proseguito con un'analisi di tipo mutation coverage. Come si può notare in dettaglio nella Figura45, tutti i mutanti risultano eliminati.

RISORSE ESTERNE

put(long, long, ByteBuf)		96%		62%
get(long, long)		100%		100%

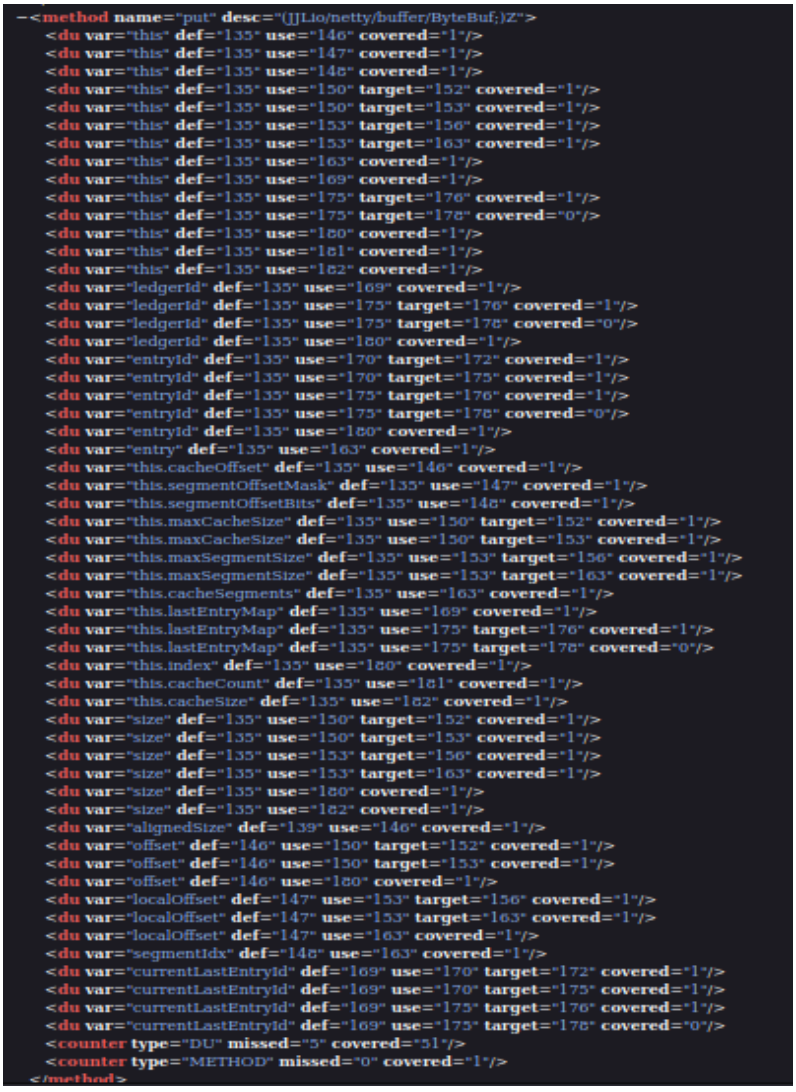
Figura1 (Coverage metodi WriteCache iniziale)

put(long, long, ByteBuf)		98%		87%
get(long, long)		100%		100%

Figura2 (Coverage metodi WriteCache finale)

```
public boolean put(long ledgerId, long entryId, ByteBuf entry) {
    int size = entry.readableBytes();

    // Align to 64 bytes so that different threads will not contend the same L1
    // cache line
    int alignedSize = align64(size);

    long offset;
    int localOffset;
    int segmentIdx;

    while (true) {
        offset = cacheOffset.getAndAdd(alignedSize);
        localOffset = (int) (offset & segmentOffsetMask);
        segmentIdx = (int) (offset >>> segmentOffsetBits);

        if ((offset + size) > maxCacheSize) {
            // Cache is full
            return false;
        } else if (maxSegmentSize - localOffset < size) {
            // If an entry is at the end of a segment, we need to get a new offset and try
            // again in next segment
            continue;
        } else {
            // Found a good offset
            break;
        }
    }

    cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());

    // Update last entryId for ledger. This logic is to handle writes for the same
    // ledger coming out of order and from different thread, though in practice it
    // should not happen and the compareAndSet should be always uncontended.
    while (true) {
        long currentLastEntryId = lastEntryMap.get(ledgerId);
        if (currentLastEntryId > entryId) {
            // A newer entry is already there
            break;
        }

        if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
            break;
        }
    }

    index.put(ledgerId, entryId, offset, size);
    cacheCount.increment();
    cacheSize.addAndGet(size);
    return true;
```

Figura3(dettaglio finale coverage metodo Put)



```
-<method name="put" desc="(JJLio/netty/buffer/ByteBuf;)Z">
  <du var="this" def="135" use="146" covered="1"/>
  <du var="this" def="135" use="147" covered="1"/>
  <du var="this" def="135" use="148" covered="1"/>
  <du var="this" def="135" use="150" target="152" covered="1"/>
  <du var="this" def="135" use="150" target="153" covered="1"/>
  <du var="this" def="135" use="153" target="156" covered="1"/>
  <du var="this" def="135" use="153" target="163" covered="1"/>
  <du var="this" def="135" use="163" covered="1"/>
  <du var="this" def="135" use="169" covered="1"/>
  <du var="this" def="135" use="175" target="176" covered="1"/>
  <du var="this" def="135" use="175" target="178" covered="0"/>
  <du var="this" def="135" use="180" covered="1"/>
  <du var="this" def="135" use="181" covered="1"/>
  <du var="this" def="135" use="182" covered="1"/>
  <du var="ledgerId" def="135" use="169" covered="1"/>
  <du var="ledgerId" def="135" use="175" target="176" covered="1"/>
  <du var="ledgerId" def="135" use="175" target="178" covered="0"/>
  <du var="ledgerId" def="135" use="180" covered="1"/>
  <du var="entryId" def="135" use="170" target="172" covered="1"/>
  <du var="entryId" def="135" use="170" target="175" covered="1"/>
  <du var="entryId" def="135" use="175" target="176" covered="1"/>
  <du var="entryId" def="135" use="175" target="178" covered="0"/>
  <du var="entryId" def="135" use="180" covered="1"/>
  <du var="entry" def="135" use="163" covered="1"/>
  <du var="this.cacheOffset" def="135" use="146" covered="1"/>
  <du var="this.segmentOffsetMask" def="135" use="147" covered="1"/>
  <du var="this.segmentOffsetBits" def="135" use="148" covered="1"/>
  <du var="this.maxCacheSize" def="135" use="150" target="152" covered="1"/>
  <du var="this.maxCacheSize" def="135" use="150" target="153" covered="1"/>
  <du var="this.maxSegmentSize" def="135" use="153" target="156" covered="1"/>
  <du var="this.maxSegmentSize" def="135" use="153" target="163" covered="1"/>
  <du var="this.cacheSegments" def="135" use="163" covered="1"/>
  <du var="this.lastEntryMap" def="135" use="169" covered="1"/>
  <du var="this.lastEntryMap" def="135" use="175" target="176" covered="1"/>
  <du var="this.lastEntryMap" def="135" use="175" target="178" covered="0"/>
  <du var="this.index" def="135" use="180" covered="1"/>
  <du var="this.cacheCount" def="135" use="181" covered="1"/>
  <du var="this.cacheSize" def="135" use="182" covered="1"/>
  <du var="size" def="135" use="150" target="152" covered="1"/>
  <du var="size" def="135" use="150" target="153" covered="1"/>
  <du var="size" def="135" use="153" target="156" covered="1"/>
  <du var="size" def="135" use="153" target="163" covered="1"/>
  <du var="size" def="135" use="180" covered="1"/>
  <du var="size" def="135" use="182" covered="1"/>
  <du var="alignedSize" def="139" use="146" covered="1"/>
  <du var="offset" def="146" use="150" target="152" covered="1"/>
  <du var="offset" def="146" use="150" target="153" covered="1"/>
  <du var="offset" def="146" use="180" covered="1"/>
  <du var="localOffset" def="147" use="153" target="156" covered="1"/>
  <du var="localOffset" def="147" use="153" target="163" covered="1"/>
  <du var="localOffset" def="147" use="163" covered="1"/>
  <du var="segmentIdx" def="148" use="163" covered="1"/>
  <du var="currentLastEntryId" def="169" use="170" target="172" covered="1"/>
  <du var="currentLastEntryId" def="169" use="170" target="175" covered="1"/>
  <du var="currentLastEntryId" def="169" use="175" target="176" covered="1"/>
  <du var="currentLastEntryId" def="169" use="175" target="178" covered="0"/>
  <counter type="DU" missed="5" covered="51"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```


Figura4(Write cache Put Data flow coverage badua)

```
134     public boolean put(long ledgerId, long entryId, ByteBuf entry) {
135         int size = entry.readableBytes();
136
137         // Align to 64 bytes so that different threads will not contend the same L1
138         // cache line
139         int alignedSize = align64(size);
140
141         long offset;
142         int localOffset;
143         int segmentIdx;
144
145         while (true) {
146             offset = cacheOffset.getAndAdd(alignedSize);
147 1           localOffset = (int) (offset & segmentOffsetMask);
148 1           segmentIdx = (int) (offset >>> segmentOffsetBits);
149
150 3           if ((offset + size) > maxCacheSize) {
151                 // Cache is full
152 1               return false;
153 2           } else if (maxSegmentSize - localOffset < size) {
154                 // If an entry is at the end of a segment, we need to get a new offset and try
155                 // again in next segment
156                 continue;
157             } else {
158                 // Found a good offset
159                 break;
160             }
161         }
162
163         cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
```

Figura5 (WriteCache Put Mutation Coverage Iniziale(1))

```
163         cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
164
165         // Update last entryId for ledger. This logic is to handle writes for the same
166         // ledger coming out of order and from different thread, though in practice it
167         // should not happen and the compareAndSet should be always uncontended.
168         while (true) {
169             long currentLastEntryId = lastEntryMap.get(ledgerId);
170 2           if (currentLastEntryId > entryId) {
171                 // A newer entry is already there
172                 break;
173             }
174
175 1           if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
176                 break;
177             }
178         }
179
180         index.put(ledgerId, entryId, offset, size);
181 1       cacheCount.increment();
182         cacheSize.addAndGet(size);
183 1       return true;
184     }
```

Figura6 (WriteCache Put Mutation Coverage Iniziale(2))

```
134     public boolean put(long ledgerId, long entryId, ByteBuf entry) {
135         int size = entry.readableBytes();
136
137         // Align to 64 bytes so that different threads will not contend the same L1
138         // cache line
139         int alignedSize = align64(size);
140
141         long offset;
142         int localOffset;
143         int segmentIdx;
144
145         while (true) {
146             offset = cacheOffset.getAndAdd(alignedSize);
147 1           localOffset = (int) (offset & segmentOffsetMask);
148 1           segmentIdx = (int) (offset >>> segmentOffsetBits);
149
150 3           if ((offset + size) > maxCacheSize) {
151                 // Cache is full
152 1               return false;
```

Figura7(Writecache Put Fix mutante 1)

```
163         cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
164
165         // Update last entryId for ledger. This logic is to handle writes for the same
166         // ledger coming out of order and from different thread, though in practice it
167         // should not happen and the compareAndSet should be always uncontended.
168         while (true) {
169             long currentLastEntryId = lastEntryMap.get(ledgerId);
170 2           if (currentLastEntryId > entryId) {
171                 // A newer entry is already there
172                 break;
173             }
174
175 1           if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
176                 break;
177             }
178         }
179
180         index.put(ledgerId, entryId, offset, size);
181 1       cacheCount.increment();
182         cacheSize.addAndGet(size);
183 1       return true;
184     }
```

Figura8(Writecache Put Fix Mutante 2)

```
    public ByteBuf get(long ledgerId, long entryId) {
        LongPair result = index.get(ledgerId, entryId);
        if (result == null) {
            return null;
        }

        long offset = result.first;
        int size = (int) result.second;
        ByteBuf entry = allocator.buffer(size, size);

        int localOffset = (int) (offset & segmentOffsetMask);
        int segmentIdx = (int) (offset >>> segmentOffsetBits);
        entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
        return entry;
    }
```

Figura9 (Dettaglio Jacoco Coverage metodo Get classe Writecache)

```
-<method name="get" desc="(JJ)Lio/netty/buffer/ByteBuf;">
   <du var="this" def="187" use="194" covered="1"/>
   <du var="this" def="187" use="196" covered="1"/>
   <du var="this" def="187" use="197" covered="1"/>
   <du var="this" def="187" use="198" covered="1"/>
   <du var="this.allocator" def="187" use="194" covered="1"/>
   <du var="this.segmentOffsetMask" def="187" use="196" covered="1"/>
   <du var="this.segmentOffsetBits" def="187" use="197" covered="1"/>
   <du var="this.cacheSegments" def="187" use="198" covered="1"/>
   <du var="result" def="187" use="188" target="189" covered="1"/>
   <du var="result" def="187" use="188" target="192" covered="1"/>
   <du var="result" def="187" use="192" covered="1"/>
   <du var="result" def="187" use="193" covered="1"/>
   <du var="result.first" def="187" use="192" covered="1"/>
   <du var="result.second" def="187" use="193" covered="1"/>
   <counter type="DU" missed="0" covered="14"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Figura10(Write cache Get Data flow coverage badua)

```
186      public ByteBuf get(long ledgerId, long entryId) {
187          LongPair result = index.get(ledgerId, entryId);
188 1        if (result == null) {
189              return null;
190          }
191
192          long offset = result.first;
193          int size = (int) result.second;
194          ByteBuf entry = allocator.buffer(size, size);
195
196 1        int localOffset = (int) (offset & segmentOffsetMask);
197 1        int segmentIdx = (int) (offset >>> segmentOffsetBits);
198          entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
199 1        return entry;
200      }
```

Figura11 (Pit Report metodo Get classe Writecache)

Figura12(Coverage iniziale metodo ComputeDigestForSendingV3 classe DigestManager)

```
private ByteBufList computeDigestAndPackageForSendingV3(long entryId, long lastAddConfirmed, long length,
                                                        ByteBuf data) {
    ByteBuf headersBuffer = Unpooled.buffer(METADATA_LENGTH + macCodeLength);
    headersBuffer.writeLong(ledgerId);
    headersBuffer.writeLong(entryId);
    headersBuffer.writeLong(lastAddConfirmed);
    headersBuffer.writeLong(length);

    int digest = update(0, headersBuffer, 0, METADATA_LENGTH);

    // don't unwrap slices
    final ByteBuf unwrapped = data.unwrap() != null && data.unwrap() instanceof CompositeByteBuf
            ? data.unwrap() : data;
    ReferenceCountUtil.retain(unwrapped);
    ReferenceCountUtil.release(data);

    if (unwrapped instanceof CompositeByteBuf) {
        CompositeByteBuf cbb = ((CompositeByteBuf) unwrapped);
        for (int i = 0; i < cbb.numComponents(); i++) {
            ByteBuf b = cbb.component(i);
            digest = update(digest, b, b.readerIndex(), b.readableBytes());
        }
    } else {
        digest = update(digest, unwrapped, unwrapped.readerIndex(), unwrapped.readableBytes());
    }
    populateValueAndReset(digest, headersBuffer);

    return ByteBufList.get(headersBuffer, unwrapped);
}
```

Figura13(Dettaglio Coverage iniziale metodo ComputeDigestForSendingV3 classe DigestManager)

Figura14(Coverage finale computeDigestAndPackageForSendingV3)

```
    private ByteBufList computeDigestAndPackageForSendingV3(long entryId, long lastAddConfirmed, long length,
                                                            ByteBuf data) {
        ByteBuf headersBuffer = Unpooled.buffer(METADATA_LENGTH + macCodeLength);
        headersBuffer.writeLong(ledgerId);
        headersBuffer.writeLong(entryId);
        headersBuffer.writeLong(lastAddConfirmed);
        headersBuffer.writeLong(length);

        int digest = update(0, headersBuffer, 0, METADATA_LENGTH);

        // don't unwrap slices
        final ByteBuf unwrapped = data.unwrap() != null && data.unwrap() instanceof CompositeByteBuf
                ? data.unwrap() : data;
        ReferenceCountUtil.retain(unwrapped);
        ReferenceCountUtil.release(data);

        if (unwrapped instanceof CompositeByteBuf) {
            CompositeByteBuf cbb = ((CompositeByteBuf) unwrapped);
            for (int i = 0; i < cbb.numComponents(); i++) {
                ByteBuf b = cbb.component(i);
                digest = update(digest, b, b.readerIndex(), b.readableBytes());
            }
        } else {
            digest = update(digest, unwrapped, unwrapped.readerIndex(), unwrapped.readableBytes());
        }
        populateValueAndReset(digest, headersBuffer);

        return ByteBufList.get(headersBuffer, unwrapped);
    }
```

Figura15(Dettaglio Coverage finale computeDigestAndPackageForSendingV3)

```
-<method name="computeDigestAndPackageForSendingV3" desc="(JJJLio/netty/buffer/ByteBuf;)Lorg/apache/bookkeeper/util/ByteBufList;">
   <du var="this" def="172" use="193" covered="1"/>
   <du var="this" def="172" use="195" covered="1"/>
   <du var="this" def="172" use="190" covered="1"/>
   <du var="data" def="172" use="181" target="181" covered="1"/>
   <du var="data" def="172" use="181" target="182" covered="1"/>
   <du var="data" def="172" use="182" covered="1"/>
   <du var="data" def="172" use="184" covered="1"/>
   <du var="data" def="172" use="181" target="181" covered="0"/>
   <du var="data" def="172" use="181" target="182" covered="1"/>
   <du var="headersBuffer" def="172" use="195" covered="1"/>
   <du var="headersBuffer" def="172" use="197" covered="1"/>
   <du var="digest" def="178" use="193" covered="1"/>
   <du var="digest" def="178" use="195" covered="0"/>
   <du var="digest" def="178" use="190" covered="1"/>
   <du var="unwrapped" def="182" use="186" target="187" covered="1"/>
   <du var="unwrapped" def="182" use="186" target="193" covered="1"/>
   <du var="unwrapped" def="182" use="193" covered="1"/>
   <du var="unwrapped" def="182" use="197" covered="1"/>
   <du var="unwrapped" def="182" use="187" covered="1"/>
   <du var="cbb" def="187" use="188" target="189" covered="1"/>
   <du var="cbb" def="187" use="188" target="192" covered="1"/>
   <du var="cbb" def="187" use="189" covered="1"/>
   <du var="i" def="188" use="188" target="189" covered="1"/>
   <du var="i" def="188" use="188" target="192" covered="0"/>
   <du var="i" def="188" use="189" covered="1"/>
   <du var="i" def="188" use="188" covered="1"/>
   <du var="digest" def="190" use="195" covered="1"/>
   <du var="digest" def="190" use="190" covered="1"/>
   <du var="i" def="188" use="188" target="189" covered="1"/>
   <du var="i" def="188" use="188" target="192" covered="1"/>
   <du var="i" def="188" use="189" covered="1"/>
   <du var="i" def="188" use="188" covered="1"/>
   <du var="digest" def="193" use="195" covered="1"/>
   <counter type="DU" missed="3" covered="30"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Figura16(Analisi Badua ComputeDigestAndPackageForSendingV3)

```
170     private ByteBufList computeDigestAndPackageForSendingV3(long entryId, long lastAddConfirmed, long length,
171                                                             ByteBuf data) {
172 1       ByteBuf headersBuffer = Unpooled.buffer(METADATA_LENGTH + macCodeLength);
173         headersBuffer.writeLong(ledgerId);
174         headersBuffer.writeLong(entryId);
175         headersBuffer.writeLong(lastAddConfirmed);
176         headersBuffer.writeLong(length);
177
178         int digest = update(0, headersBuffer, 0, METADATA_LENGTH);
179
180         // don't unwrap slices
181 2       final ByteBuf unwrapped = data.unwrap() != null && data.unwrap() instanceof CompositeByteBuf
182                 ? data.unwrap() : data;
183         ReferenceCountUtil.retain(unwrapped);
184         ReferenceCountUtil.release(data);
185
186 1       if (unwrapped instanceof CompositeByteBuf) {
187             CompositeByteBuf cbb = ((CompositeByteBuf) unwrapped);
188 2           for (int i = 0; i < cbb.numComponents(); i++) {
189                 ByteBuf b = cbb.component(i);
190                 digest = update(digest, b, b.readerIndex(), b.readableBytes());
191             }
192         } else {
193             digest = update(digest, unwrapped, unwrapped.readerIndex(), unwrapped.readableBytes());
194         }
195 1       populateValueAndReset(digest, headersBuffer);
196
197 1       return ByteBufList.get(headersBuffer, unwrapped);
198     }
```

Figura17(Analisi Pit ComputeDigestAndPackageForSendingV3)

```
243.    private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
244.            throws BKDigestMatchException {
245.
246.        if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
247.            logger.error("Data received is smaller than the minimum for this digest type. "
248.                    + " Either the packet it corrupt, or the wrong digest is configured. "
249.                    + " Digest type: {}, Packet Length: {}",
250.                    this.getClass().getName(), dataReceived.readableBytes());
251.            throw new BKDigestMatchException();
252.        }
253.        int digest = update(0, dataReceived, 0, METADATA_LENGTH);
254.
255.        int offset = METADATA_LENGTH + macCodeLength;
256.        digest = update(digest, dataReceived, offset, dataReceived.readableBytes() - offset);
257.
258.        if (isInt32Digest()) {
259.            int receivedDigest = dataReceived.getInt(METADATA_LENGTH);
260.            if (receivedDigest != digest) {
261.                logger.error("Digest mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
262.                throw new BKDigestMatchException();
263.            }
264.        } else {
265.            ByteBuf digestBuf = DIGEST_BUFFER.get();
266.            digestBuf.clear();
267.            populateValueAndReset(digest, digestBuf);
268.
269.            if (!ByteBufUtil.equals(digestBuf, 0, dataReceived, METADATA_LENGTH, macCodeLength)) {
270.                logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
271.                throw new BKDigestMatchException();
272.            }
273.        }
274.
275.        long actualLedgerId = dataReceived.readLong();
276.        long actualEntryId = dataReceived.readLong();
277.
278.        if (actualLedgerId != ledgerId) {
279.            logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
280.                    + actualLedgerId);
281.            throw new BKDigestMatchException();
282.        }
283.
284.        if (!skipEntryIdCheck && actualEntryId != entryId) {
285.            logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
286.                    + actualEntryId);
287.            throw new BKDigestMatchException();
288.        }
289.
290.    }
```

Figura18(Dettaglio Coverage iniziale metodo verifyDigest)

verifyDigest(long, ByteBuf, boolean)	100%	100%
verifyDigestAndReturnData(long, ByteBuf)	100%	n/a
verifyDigestAndReturnLastConfirmed(ByteBuf)	100%	n/a

Figura 19,20 (Coverage Finale VerifyDigest+metodipubblici)

```
243.    private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
244.            throws BKDigestMatchException {
245.
246.        if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
247.            logger.error("Data received is smaller than the minimum for this digest type. "
248.                    + " Either the packet it corrupt, or the wrong digest is configured. "
249.                    + " Digest type: {}, Packet Length: {}",
250.                    this.getClass().getName(), dataReceived.readableBytes());
251.            throw new BKDigestMatchException();
252.        }
253.        int digest = update(0, dataReceived, 0, METADATA_LENGTH);
254.
255.        int offset = METADATA_LENGTH + macCodeLength;
256.        digest = update(digest, dataReceived, offset, dataReceived.readableBytes() - offset);
257.
258.        if (isInt32Digest()) {
259.            int receivedDigest = dataReceived.getInt(METADATA_LENGTH);
260.            if (receivedDigest != digest) {
261.                logger.error("Digest mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
262.                throw new BKDigestMatchException();
263.            }
264.        } else {
265.            ByteBuf digestBuf = DIGEST_BUFFER.get();
266.            digestBuf.clear();
267.            populateValueAndReset(digest, digestBuf);
268.
269.            if (!ByteBufUtil.equals(digestBuf, 0, dataReceived, METADATA_LENGTH, macCodeLength)) {
270.                logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
271.                throw new BKDigestMatchException();
272.            }
273.        }
274.
275.        long actualLedgerId = dataReceived.readLong();
276.        long actualEntryId = dataReceived.readLong();
277.
278.        if (actualLedgerId != ledgerId) {
279.            logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
280.                    + actualLedgerId);
281.            throw new BKDigestMatchException();
282.        }
283.
284.        if (!skipEntryIdCheck && actualEntryId != entryId) {
285.            logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
286.                    + actualEntryId);
287.            throw new BKDigestMatchException();
288.        }
289.
290.    }
```

Figura 21 (Dettaglio Coverage finale VerifyDigest)

```
-<method name="verifyDigest" desc="(JLio/netty/buffer/ByteBuf;Z)V">
   <du var="this" def="246" use="246" target="247" covered="1"/>
   <du var="this" def="246" use="246" target="253" covered="1"/>
   <du var="this" def="246" use="253" covered="1"/>
   <du var="this" def="246" use="255" covered="1"/>
   <du var="this" def="246" use="256" covered="1"/>
   <du var="this" def="246" use="258" target="259" covered="1"/>
   <du var="this" def="246" use="258" target="265" covered="1"/>
   <du var="this" def="246" use="267" covered="1"/>
   <du var="this" def="246" use="269" target="270" covered="1"/>
   <du var="this" def="246" use="269" target="275" covered="1"/>
   <du var="this" def="246" use="278" target="279" covered="1"/>
   <du var="this" def="246" use="278" target="284" covered="1"/>
   <du var="this" def="246" use="279" covered="1"/>
   <du var="this" def="246" use="270" covered="1"/>
   <du var="this" def="246" use="261" covered="1"/>
   <du var="this" def="246" use="247" covered="1"/>
   <du var="entryId" def="246" use="284" target="285" covered="1"/>
   <du var="entryId" def="246" use="284" target="290" covered="1"/>
   <du var="entryId" def="246" use="285" covered="1"/>
   <du var="entryId" def="246" use="270" covered="1"/>
   <du var="entryId" def="246" use="261" covered="1"/>
   <du var="dataReceived" def="246" use="246" target="247" covered="1"/>
   <du var="dataReceived" def="246" use="246" target="253" covered="1"/>
   <du var="dataReceived" def="246" use="253" covered="1"/>
   <du var="dataReceived" def="246" use="256" covered="1"/>
   <du var="dataReceived" def="246" use="269" target="270" covered="1"/>
   <du var="dataReceived" def="246" use="269" target="275" covered="1"/>
   <du var="dataReceived" def="246" use="275" covered="1"/>
   <du var="dataReceived" def="246" use="276" covered="1"/>
   <du var="dataReceived" def="246" use="259" covered="1"/>
   <du var="dataReceived" def="246" use="247" covered="1"/>
   <du var="skipEntryIdCheck" def="246" use="284" target="284" covered="1"/>
   <du var="skipEntryIdCheck" def="246" use="284" target="290" covered="1"/>
   <du var="this.macCodeLength" def="246" use="246" target="247" covered="1"/>
```

```
<du var="dataReceived" def="246" use="247" covered="1"/>
<du var="skipEntryIdCheck" def="246" use="284" target="284" covered="1"/>
<du var="skipEntryIdCheck" def="246" use="284" target="290" covered="1"/>
<du var="this.macCodeLength" def="246" use="246" target="247" covered="1"/>
<du var="this.macCodeLength" def="246" use="246" target="253" covered="1"/>
<du var="this.macCodeLength" def="246" use="255" covered="1"/>
<du var="this.macCodeLength" def="246" use="269" target="270" covered="1"/>
<du var="this.macCodeLength" def="246" use="269" target="275" covered="1"/>
<du var="logger" def="246" use="285" covered="1"/>
<du var="logger" def="246" use="279" covered="1"/>
<du var="logger" def="246" use="270" covered="1"/>
<du var="logger" def="246" use="261" covered="1"/>
<du var="logger" def="246" use="247" covered="1"/>
<du var="this.ledgerId" def="246" use="278" target="279" covered="1"/>
<du var="this.ledgerId" def="246" use="278" target="284" covered="1"/>
<du var="this.ledgerId" def="246" use="279" covered="1"/>
<du var="this.ledgerId" def="246" use="270" covered="1"/>
<du var="this.ledgerId" def="246" use="261" covered="1"/>
<du var="DIGEST_BUFFER" def="246" use="265" covered="1"/>
<du var="digest" def="256" use="267" covered="1"/>
<du var="digest" def="256" use="260" target="261" covered="1"/>
<du var="digest" def="256" use="260" target="264" covered="1"/>
<du var="receivedDigest" def="259" use="260" target="261" covered="1"/>
<du var="receivedDigest" def="259" use="260" target="264" covered="1"/>
<du var="digestBuf" def="265" use="269" target="270" covered="1"/>
<du var="digestBuf" def="265" use="269" target="275" covered="1"/>
<du var="actualLedgerId" def="275" use="278" target="279" covered="1"/>
<du var="actualLedgerId" def="275" use="278" target="284" covered="1"/>
<du var="actualLedgerId" def="275" use="279" covered="1"/>
<du var="actualEntryId" def="276" use="284" target="285" covered="1"/>
<du var="actualEntryId" def="276" use="284" target="290" covered="1"/>
<du var="actualEntryId" def="276" use="285" covered="1"/>
<counter type="DU" missed="0" covered="62"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 22,23 (Data Flow Coverage Verify Digest)

```
243     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
244             throws BKDigestMatchException {
245
246 3       if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
247             logger.error("Data received is smaller than the minimum for this digest type. "
248                     + " Either the packet it corrupt, or the wrong digest is configured. "
249                     + " Digest type: {}, Packet Length: {}",
250                     this.getClass().getName(), dataReceived.readableBytes());
251             throw new BKDigestMatchException();
252         }
253         int digest = update(0, dataReceived, 0, METADATA_LENGTH);
254
255 1       int offset = METADATA_LENGTH + macCodeLength;
256 1       digest = update(digest, dataReceived, offset, dataReceived.readableBytes() - offset);
257
258 1       if (isInt32Digest()) {
259             int receivedDigest = dataReceived.getInt(METADATA_LENGTH);
260 1           if (receivedDigest != digest) {
261                 logger.error("Digest mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
262                 throw new BKDigestMatchException();
263             }
264         } else {
265             ByteBuf digestBuf = DIGEST_BUFFER.get();
266             digestBuf.clear();
267 1           populateValueAndReset(digest, digestBuf);
268
269 1           if (!ByteBufUtil.equals(digestBuf, 0, dataReceived, METADATA_LENGTH, macCodeLength)) {
270                 logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
271                 throw new BKDigestMatchException();
272             }
273         }
274
275         long actualLedgerId = dataReceived.readLong();
276         long actualEntryId = dataReceived.readLong();
277
278 1       if (actualLedgerId != ledgerId) {
279             logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
280                     + actualLedgerId);
281             throw new BKDigestMatchException();
282         }
283
284 2       if (!skipEntryIdCheck && actualEntryId != entryId) {
285             logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
286                     + actualEntryId);
287             throw new BKDigestMatchException();
288         }
289
290     }
```

Figura24(Verify Digest Final Mutation Coverage)

Figura25(percentuali coverage encode)

```
94.    public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
95.            throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
96.            IllegalBlockSizeException, BadPaddingException {
97.
98.        String encoded = null;
99.
100.       if (value != null) {
101.           if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
102.               Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
103.               cipher.init(Cipher.ENCRYPT_MODE, keySpec);
104.
105.               encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
106.           } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
107.               encoded = BCrypt.hashpw(value, BCrypt.gensalt());
108.           } else {
109.               encoded = getDigester(cipherAlgorithm).digest(value);
110.           }
111.       }
112.
113.       return encoded;
114.   }
115.
```

Figura26(dettaglio coverage Encode)

```
94      public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
95              throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
96              IllegalBlockSizeException, BadPaddingException {
97
98          String encoded = null;
99
100 1       if (value != null) {
101 2           if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
102                 Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
103 1               cipher.init(Cipher.ENCRYPT_MODE, keySpec);
104
105                 encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
106 1           } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
107                 encoded = BCrypt.hashpw(value, BCrypt.gensalt());
108             } else {
109                 encoded = getDigester(cipherAlgorithm).digest(value);
110             }
111         }
112
113 1       return encoded;
114     }
```

Figura27(dettaglio mutation coverage Encode)

Figura 28(Coverage JaCoCo metodo verify)

```
116.    public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
117.        boolean verified = false;
118.
119.        try {
120.            if (value != null) {
121.                if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
122.                    verified = encode(value, cipherAlgorithm).equals(encoded);
123.                } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
124.                    verified = BCrypt.checkpw(value, encoded);
125.                } else {
126.                    verified = getDigester(cipherAlgorithm).matches(value, encoded);
127.                }
128.            }
129.        } catch (Exception e) {
130.            LOG.error("Could not verify encoded value", e);
131.        }
132.
133.        return verified;
134.    }
```

Figura29(Dettaglio coverage metodo Verify)

```
116     public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
117         boolean verified = false;
118
119         try {
120 1           if (value != null) {
121 2               if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
122                     verified = encode(value, cipherAlgorithm).equals(encoded);
123 1               } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
124                     verified = BCrypt.checkpw(value, encoded);
125                 } else {
126                     verified = getDigester(cipherAlgorithm).matches(value, encoded);
127                 }
128             }
129         } catch (Exception e) {
130             LOG.error("Could not verify encoded value", e);
131         }
132
133 2       return verified;
134     }
```

Figura30(Dettaglio mutation verify)

decode(String, CipherAlgorithm)		100%		100%

Figura31(Coverage metodo decode)

```
136.    public String decode(final String encoded, final CipherAlgorithm cipherAlgorithm)
137.            throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
138.            IllegalBlockSizeException, BadPaddingException {
139.
140.        String decoded = null;
141.
142.        if (encoded != null && cipherAlgorithm == CipherAlgorithm.AES) {
143.            Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
144.            cipher.init(Cipher.DECRYPT_MODE, keySpec);
145.
146.            decoded = new String(cipher.doFinal(Base64.getDecoder().decode(encoded)), StandardCharsets.UTF_8);
147.        }
148.
149.        return decoded;
150.    }
151
```

Figura32(Dettaglio coverage metodo decode)

```
136     public String decode(final String encoded, final CipherAlgorithm cipherAlgorithm)
137             throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
138             IllegalBlockSizeException, BadPaddingException {
139
140         String decoded = null;
141
142 2       if (encoded != null && cipherAlgorithm == CipherAlgorithm.AES) {
143             Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
144 1           cipher.init(Cipher.DECRYPT_MODE, keySpec);
145
146             decoded = new String(cipher.doFinal(Base64.getDecoder().decode(encoded)), StandardCharsets.UTF_8);
147         }
148
149 1       return decoded;
150     }
```

Figura33(Dettaglio Mutation metodo Decode)

Encryptor

Element	Missed Instructions	Cov.	Missed Branches	Cov.
Encryptor(String)		50%		50%
getInstance(String)		96%		75%
getDigester(CipherAlgorithm)		100%		100%
encode(String, CipherAlgorithm)		100%		100%
verify(String, CipherAlgorithm, String)		100%		100%
decode(String, CipherAlgorithm)		100%		100%
static {...}		100%		n/a
getInstance()		100%		n/a
Total	29 of 280	89%	2 of 30	93%

Figura34(Coverage classe Encryptor prima dei miglioramenti)

```
71.     private Encryptor(final String secretKey) {
72.         String actualKey = secretKey;
73.         if (actualKey.length() < 16) {
74.             StringBuilder actualKeyPadding = new StringBuilder(actualKey);
75.             int length = 16 - actualKey.length();
76.             String randomChars = SecureRandomUtils.generateRandomPassword(length);
77.
78.             actualKeyPadding.append(randomChars);
79.             actualKey = actualKeyPadding.toString();
80.             LOG.warn("The secret key is too short (< 16), adding some random characters. "
81.                     + "Passwords encrypted with AES and this key will not be recoverable "
82.                     + "as a result if the container is restarted.");
83.         }
84.
85.         try {
86.             keySpec = new SecretKeySpec(ArrayUtils.subarray(
87.                     actualKey.getBytes(StandardCharsets.UTF_8), 0, 16),
88.                     CipherAlgorithm.AES.getAlgorithm());
89.         } catch (Exception e) {
90.             LOG.error("Error during key specification", e);
91.         }
92.     }
93.
```

Figura35(dettaglio coverage mancante classe Encryptor)

Encryptor	98%	100%

Figura36(Coverage finale totale classe Encryptor)

Figura37(generate coverage)

```
82.     protected DefaultPasswordRuleConf merge(final List<DefaultPasswordRuleConf> defaultRuleConfs) {
83.         DefaultPasswordRuleConf result = new DefaultPasswordRuleConf();
84.         result.setMinLength(VERY_MIN_LENGTH);
85.         result.setMaxLength(VERY_MAX_LENGTH);
86.
87.         defaultRuleConfs.forEach(ruleConf -> {
88.             if (ruleConf.getMinLength() > result.getMinLength()) {
89.                 result.setMinLength(ruleConf.getMinLength());
90.             }
91.
92.             if (ruleConf.getMaxLength() > 0 && ruleConf.getMaxLength() < result.getMaxLength()) {
93.                 result.setMaxLength(ruleConf.getMaxLength());
94.             }
95.
96.             if (ruleConf.getAlphabetical() > result.getAlphabetical()) {
97.                 result.setAlphabetical(ruleConf.getAlphabetical());
98.             }
99.
100.            if (ruleConf.getUppercase() > result.getUppercase()) {
101.                result.setUppercase(ruleConf.getUppercase());
102.            }
103.
104.            if (ruleConf.getLowercase() > result.getLowercase()) {
105.                result.setLowercase(ruleConf.getLowercase());
106.            }
107.
108.            if (ruleConf.getDigit() > result.getDigit()) {
109.                result.setDigit(ruleConf.getDigit());
110.            }
111.
112.            if (ruleConf.getSpecial() > result.getSpecial()) {
113.                result.setSpecial(ruleConf.getSpecial());
114.            }
115.
116.            if (!ruleConf.getSpecialChars().isEmpty()) {
117.                result.getSpecialChars().addAll(ruleConf.getSpecialChars().stream().
```

Figura38(Coverage iniziale Merge1)

```
15.
16.             if (!ruleConf.getSpecialChars().isEmpty()) {
17.                 result.getSpecialChars().addAll(ruleConf.getSpecialChars().stream().
18.                         filter(c -> !result.getSpecialChars().contains(c)).collect(Collectors.toList()));
19.             }
20.
21.             if (!ruleConf.getIllegalChars().isEmpty()) {
22.                 result.getIllegalChars().addAll(ruleConf.getIllegalChars().stream().
23.                         filter(c -> !result.getIllegalChars().contains(c)).collect(Collectors.toList()));
24.             }
25.
26.             if (ruleConf.getRepeatSame() > result.getRepeatSame()) {
27.                 result.setRepeatSame(ruleConf.getRepeatSame());
28.             }
29.
30.             if (!result.isUsernameAllowed()) {
31.                 result.setUsernameAllowed(ruleConf.isUsernameAllowed());
32.             }
33.
34.             if (!ruleConf.getWordsNotPermitted().isEmpty()) {
35.                 result.getWordsNotPermitted().addAll(ruleConf.getWordsNotPermitted().stream().
36.                         filter(w -> !result.getWordsNotPermitted().contains(w)).collect(Collectors.toList()));
37.             }
38.         });
39.
40.         if (result.getMinLength() == 0) {
41.             result.setMinLength(
42.                     result.getMaxLength() < MIN_LENGTH_IF_ZERO ? result.getMaxLength() : MIN_LENGTH_IF_ZERO);
43.         }
44.         if (result.getMinLength() > result.getMaxLength()) {
45.             result.setMaxLength(result.getMinLength());
46.         }
47.
48.         return result;
49.     }
```

Figura39(Coverage Iniziale Merge2)

```
82.     protected DefaultPasswordRuleConf merge(final List<DefaultPasswordRuleConf> defaultRuleConfs) {
83.         DefaultPasswordRuleConf result = new DefaultPasswordRuleConf();
84.         result.setMinLength(VERY_MIN_LENGTH);
85.         result.setMaxLength(VERY_MAX_LENGTH);
86.
87.         defaultRuleConfs.forEach(ruleConf -> {
88.             if (ruleConf.getMinLength() > result.getMinLength()) {
89.                 result.setMinLength(ruleConf.getMinLength());
90.             }
91.
92.             if (ruleConf.getMaxLength() > 0 && ruleConf.getMaxLength() < result.getMaxLength()) {
93.                 result.setMaxLength(ruleConf.getMaxLength());
94.             }
95.
96.             if (ruleConf.getAlphabetical() > result.getAlphabetical()) {
97.                 result.setAlphabetical(ruleConf.getAlphabetical());
98.             }
99.
100.            if (ruleConf.getUppercase() > result.getUppercase()) {
101.                result.setUppercase(ruleConf.getUppercase());
102.            }
103.
104.            if (ruleConf.getLowercase() > result.getLowercase()) {
105.                result.setLowercase(ruleConf.getLowercase());
106.            }
107.
108.            if (ruleConf.getDigit() > result.getDigit()) {
109.                result.setDigit(ruleConf.getDigit());
110.            }
111.
112.            if (ruleConf.getSpecial() > result.getSpecial()) {
113.                result.setSpecial(ruleConf.getSpecial());
114.            }
115.
116.            if (!ruleConf.getSpecialChars().isEmpty()) {
117.                result.getSpecialChars().addAll(ruleConf.getSpecialChars().stream()
```

Figura40(Coverage finale merge1)

```
115.
116.            if (!ruleConf.getSpecialChars().isEmpty()) {
117.                result.getSpecialChars().addAll(ruleConf.getSpecialChars().stream().
118.                        filter(c -> !result.getSpecialChars().contains(c)).collect(Collectors.toList()));
119.            }
120.
121.            if (!ruleConf.getIllegalChars().isEmpty()) {
122.                result.getIllegalChars().addAll(ruleConf.getIllegalChars().stream().
123.                        filter(c -> !result.getIllegalChars().contains(c)).collect(Collectors.toList()));
124.            }
125.
126.            if (ruleConf.getRepeatSame() > result.getRepeatSame()) {
127.                result.setRepeatSame(ruleConf.getRepeatSame());
128.            }
129.
130.            if (!result.isUsernameAllowed()) {
131.                result.setUsernameAllowed(ruleConf.isUsernameAllowed());
132.            }
133.
134.            if (!ruleConf.getWordsNotPermitted().isEmpty()) {
135.                result.getWordsNotPermitted().addAll(ruleConf.getWordsNotPermitted().stream().
136.                        filter(w -> !result.getWordsNotPermitted().contains(w)).collect(Collectors.toList()));
137.            }
138.        });
139.
140.        if (result.getMinLength() == 0) {
141.            result.setMinLength(
142.                    result.getMaxLength() < MIN_LENGTH_IF_ZERO ? result.getMaxLength() : MIN_LENGTH_IF_ZERO);
143.        }
144.        if (result.getMinLength() > result.getMaxLength()) {
145.            result.setMaxLength(result.getMinLength());
146.        }
147.
148.        return result;
149.    }
```

Figura41(Coverage finale merge2)

```
53.     public String generate(final ExternalResource resource) {
54.         List<PasswordPolicy> policies = new ArrayList<>();
55.
56.         if (resource.getPasswordPolicy() != null) {
57.             policies.add(resource.getPasswordPolicy());
58.         }
59.
60.         return generate(policies);
61.     }
```

Figura42(mancata coverage generate from external resource)

```
53.     public String generate(final ExternalResource resource) {
54.         List<PasswordPolicy> policies = new ArrayList<>();
55.
56.         if (resource.getPasswordPolicy() != null) {
57.             policies.add(resource.getPasswordPolicy());
58.         }
59.
60.         return generate(policies);
61.     }
```

Figura43(Coverage finale generate from external resource)

DefaultPasswordGenerator	99%	88%

Figura44 (Coverage finale DefaultPasswordGenerator)

```
53      public String generate(final ExternalResource resource) {
54          List<PasswordPolicy> policies = new ArrayList<>();
55
56  1       if (resource.getPasswordPolicy() != null) {
57              policies.add(resource.getPasswordPolicy());
58          }
59
60  1       return generate(policies);
61      }
62
63      @Override
64      public String generate(final List<PasswordPolicy> policies) {
65          List<DefaultPasswordRuleConf> ruleConfs = new ArrayList<>();
66
67  2       policies.stream().forEach(policy -> policy.getRules().forEach(impl -> {
68              try {
69  1               ImplementationManager.buildPasswordRule(impl).ifPresent(rule -> {
70  1                   if (rule.getConf() instanceof DefaultPasswordRuleConf) {
71                          ruleConfs.add((DefaultPasswordRuleConf) rule.getConf());
72                      }
73                  });
74              } catch (Exception e) {
75                  LOG.error("Invalid {}, ignoring...", impl, e);
76              }
77          });
78
79  1       return generate(merge(ruleConfs));
80      }
```

Figura45 (Dettaglio Mutation Coverage generate)