Trusted Server

|  |  |  |
| --- | --- | --- |
| T\_User | | |
| **Field** | **Example** | **Notes** |
| UserID | 1 |  |
| FirstName | Super |  |
| LastName | User |  |
| EmailAddress | user@provenance.com |  |
| Password | sha1:1000:mIpUW5aRzopQdn/XFRZruFqR4prztY6l:N9PUmPUZoFUEAK/ZPG6MSxHwq5oORedp | Salted and hashed in PasswordHash.java using PBKDF2WithHmacSHA1 |
| SslKey | -----BEGIN RSA PRIVATE KEY-----  …  -----END RSA PRIVATE KEY----- | Prepared in TrustedLocalSQLiteDataAccessor.java using an X509CertificateGenerator by calling getPEMPrivateKey and used for secure communication |
| SslCertificate | -----BEGIN CERTIFICATE-----  …  -----END CERTIFICATE----- | Prepared in TrustedLocalSQLiteDataAccessor.java using an X509CertificateGenerator by calling getPEMCertificate and used for secure communication |
| PrivateKey | Binary data | Prepared in TrustedLocalSQLiteDataAccessor.java using an X509CertificateGenerator by calling getPrivateKey |
| PublicKey | Binary data | Prepared in TrustedLocalSQLiteDataAccessor.java using an X509CertificateGenerator by calling getPublicKey |
| IsActive | TRUE |  |
| UserTypeID | 1 | EDITOR(1),  AUDITOR(2),  ADMINISTRATOR(3),  PROGRAMMER(4),  UNKNOWN(-1) |

|  |  |  |
| --- | --- | --- |
| T\_Document | | |
| **Field** | **Example** | **Notes** |
| DocumentID | 1 |  |
| Title | Hello, world! |  |
| LatestVersionID | 69623a4a-79c0-411f-b5bc-1f7473979300 | Duplicate of T\_DocumentVersion.VersionID |
| VersionNo | 4 | Duplicate of T\_DocumentVersion.VersionNo |
| UpdatedBy | 4 | Maps to T\_User.UserID |
| UpdatedOn | 2015-09-08 20:35:49.690 |  |
| CreatedBy | 4 | Maps to T\_User.UserID |
| CreatedOn | 2015-09-08 18:16:34.901 |  |
| UniqueKey | 14CgTzstsGe6Pz5wpximUQ== | Prepared in KeyGenerator.java using AES with keysize 128 |

|  |  |  |
| --- | --- | --- |
| T\_UserDocumentMapping | | |
| **Field** | **Example** | **Notes** |
| UserID | 4 | Maps to T\_User.UserID |
| DocumentID | 1 | Maps to T\_Document.DocumentID |
| IsEditor | TRUE |  |
| IsAuditor | TRUE |  |
| IsAdministrator | FALSE |  |

|  |  |  |
| --- | --- | --- |
| T\_DocumentVersion | | |
| **Field** | **Example** | **Notes** |
| VersionID | 69623a4a-79c0-411f-b5bc-1f7473979300 | Prepared in TrustedLocalSQLiteDataAccessor.java using an UUID by randomUUID() |
| PrevVersionID | bcf0fb4b-c1a9-4ab3-bfe3-0eb17954a9bc |  |
| DocumentID | 1 | Maps to T\_Document.DocumentID |
| UpdatedBy | 4 | Maps to T\_User.UserID |
| UpdatedOn | 2015-09-08 20:35:49.690 |  |
| VersionNo | 4 | Incremental number |

|  |  |  |
| --- | --- | --- |
| T\_Access | | |
| **Field** | **Example** | **Notes** |
| UserID | 4 | Maps to T\_User.UserID |
| DocumentID | 1 | Duplicate of T\_Document.DocumentID |
| VersionID | 69623a4a-79c0-411f-b5bc-1f7473979300 | Maps to T\_DocumentVersion.VersionID |
| VersionNo | 4 | Duplicate of T\_DocumentVersion.VersionNo |
| AccessedOn | 2015-09-08 20:35:49.492 |  |
| AccessTypeID | 3 | NEW(1),  OPEN(2),  SAVE(3),  AUDIT(4),  UNKNOWN(-1) |
| Round | 0 | 0 or 1 |

Provenance Server

|  |  |  |
| --- | --- | --- |
| P\_DocumentBucket | | |
| **Field** | **Example** | **Notes** |
| VersionID | 69623a4a-79c0-411f-b5bc-1f7473979300 | Maps to T\_DocumentVersion.VersionID |
| Document | bRfL75+cbLxQ+Sg46HAHZHLtiGQwj0lUzexcucSJxnimcK1N4UXgIIt+CF2kEaT3uITFwRRj… | Encrypted with T\_Document.UniqueKey |

|  |  |  |
| --- | --- | --- |
| P\_ProvenanceBucket | | |
| **Field** | **Example** | **Notes** |
| VersionID | 69623a4a-79c0-411f-b5bc-1f7473979300 | Maps to T\_DocumentVersion.VersionID |
| PrevVersionID | bcf0fb4b-c1a9-4ab3-bfe3-0eb17954a9bc | Maps to T\_DocumentVersion.VersionID |
| Provenance | ["{\n\t\"pi\":\t\"user@provenance.com\",\n\t\"e\":\t\"mqHGnCF3KXrYFZit2xNb7eJb/mQm87uz/fcqw0bUV211ruHiMAir/YSpuBHvDY/Zs6Bpc/Lt\\nbUHKH9rozPkcs2y0oSqxUpZTHUcKfP4Tfy2Ngg/hNyPKqnwjHFkl7zVrmbi9sPc4xoQeiS2+\\n5JUviD+WkwfQ420WoHRrzOmk/hspZQ9lOpO3b2OTHWG6H4uvfb8Z/ywUSodrmEywj2kyPXcM… | The example to the left is not encrypted, but the field has since been encrypted using the same T\_Document UniqueKey that is used for the document.  This does mean double encryption of the Provenance events. |

|  |  |  |
| --- | --- | --- |
| P\_TriggerTable | | |
| **Field** | **Example** | **Notes** |
| VersionID | 69623a4a-79c0-411f-b5bc-1f7473979300 | See P\_ProvenanceBucket |
| PrevVersionID | bcf0fb4b-c1a9-4ab3-bfe3-0eb17954a9bc | See P\_ProvenanceBucket |
| Provenance | ["{\n\t\"pi\":\t\"user@provenance.com\",\n\t\"e\":\t\"mqHGnCF3KXrYFZit2xNb7eJb/mQm87uz/fcqw0bUV211ruHiMAir/YSpuBHvDY/Zs6Bpc/Lt\\nbUHKH9rozPkcs2y0oSqxUpZTHUcKfP4Tfy2Ngg/hNyPKqnwjHFkl7zVrmbi9sPc4xoQeiS2+\\n5JUviD+WkwfQ420WoHRrzOmk/hspZQ9lOpO3b2OTHWG6H4uvfb8Z/ywUSodrmEywj2kyPXcM… | See P\_ProvenanceBucket |
| Index | 5 | This is used to sort Provenance entries when multiple users have edited a document. The chain of VersionIDs and PrevVersionIDs is built downward from the current version to the first version using data from P\_ProvenanceBucket and triggers. [Provenance is redundant and could be fetched with a join.] |

Provenance Entry

provenances: [

[ // entries from first save

{ // first entry

pi: user@provenance.com, // principal identifier

e: “...”, // event, encrypted XML, see below, ~1314 bytes

dh: “E3B0...”, // document hash of 64 bytes

cs: “NUUy...”, // checksum, ~328 bytes

pk: “MIGf...”, // public key, ~216 bytes

sz: 162, // public key size

km: [ // keying material

{ // first auditor

id: 4, // userId

sk: “Hwdo...”, // session key ~178 bytes

sz: 16

},

{ // second auditor

} ... / additional auditors

]

},

{ // second entry

}... // more entries

],

[ // entries from second save

{ // first entry

}... // more entries

]... // entries from more saves

]

Provenance Event

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!DOCTYPE boost\_serialization>

<boost\_serialization signature="serialization::archive" version="10">

<ev class\_id="0"

class\_name="com::haathi::provenance::InsertTextProvenance"

tracking\_level="1" version="0" object\_id="\_0">

<pc class\_id="2" tracking\_level="0" version="0">

<ts>1-2345/6/7 12:34:56.789</ts>

<pi>user@provenance.com</pi>

</pc>

<pos>0</pos>

<txt>T</txt>

</ev>

</boost\_serialization>

Event Encryption Algorithm

sessionKey := <128-bit AES key from Trusted server>

publicKeyu := <1024-bit RSA key from TrustedServer>

principalIdentifieru := <user@provenance.com> // email address of current user

eventi, i > 0 := {principalIdentifieru, timestampi, eventDetailsi}

encryptedEventi, i > 0 := base64(encryptAES,ECB,sessionKey(serializeXML(eventi)))

documentStatei, i > 0 := <complete text of document after eventi>

documentHashi, i > 0 := hex(hashSHA256(documentStatei))

keyingMaterialj := {userIdj, base64(encryptRSAES\_OAEP\_SHA,publicKeyj(sessionKey)), publicKeySizej}

keyingMaterials := serializeJSON([keyingMaterialj = 1, …, keyingMaterialj = n]) // for every sharer

checksum0 := “”

checksumMateriali, i > 0 := {principalIdentifieru, encryptedEventi, documentHashi, publicKeyu, keyingMaterials}

checksumi, i > 0 := hex(hashSHA256(hex(hashSHA256(checksumMateriali)) | checksumi-1))

entryi, i > 0 := serializeJSON({principalIdentifieru, encryptedEventi, documentHashi, publicKeyu, keyingMaterials, checksumi, publicKeySize})

[Highlighting has been added to show data originating from the Trusted server, the potentially hacked Haathi client~~, and from the server but unnecessarily processed by the client~~.]

Watermark Embedding Algorithm

Let Documents = {D1..d} be the set of all documents in the system

Let Dij be version j of document Di

Let DocumentVersions = {Di..d1..v} be the set of all versions of all documents

Let Users = {U1..u} be the set of all users of the system

Let Dij •Uk be an interaction between document version Dij and user Uk

Let Interactions = {D1..d1..v •U1..u} be the set of “meetings” between document versions and users

Let D'ij be the watermarked document version Dij

Let ID(Di) be the unique identifier for document Di

Let ID(Uk) be the unique identifier for user Uk

Let V(Dij) be the unique identifier for document version Dij

Let PK(Uk) be the private RSA key for user Uk

PRNG is a Pseudo Random Number Generator

Let G(KG, CG, DG), the generator, be the tuple of

KernelPRNG KG, an ANSI X9.17 generator using AES

ChaffPRNG CG, an ANSI X9.17 generator using DES\_EDE3, and

DecisionPRNG DG, a linear congruential generator

algorithm newGenerator

input: ID(Di), V(Dij), PK(Uk), ID(Uk)

output: G(WG, CG, DG)

digest = SHA256(PK(Uk)||ID(Di)||ID(Uk))

seed = V(Dij)

KG = new KernelPRNG(seed, digest)

CG = new ChaffPRNG(seed, digest)

DG = new DecisionPRNG(seed)

return G(WG, CG, DG)

Furthermore,

Let [b1, b2, …, b64] be a list of watermark bits

Let M([b1, b2, …, b64], i) be a watermark scheme with a list of watermark bits for application at index i into the bits. In other words, it next wants to apply bi, having already applied b1,…,bi-1. i is initialized to 65 even though no bits have been applied.

Let Marks = {M1..m} be the set of watermarking schemes available

algorithm GenerateWatermark

input: G(KG, CG, DG), the generator

output: [b1, b2, …, b64], the watermark bits

if decision generator DG says so then

return [b1, b2, …, b64] = random bits from kernel generator WG

else

return [b1, b2, …, b64] = random bits from chaff generator CG

algorithm MarkObject

input: watermark method m, object o, generator G

for as many watermark bits as can be applied to o by m

if the index i of m is > 64

set watermark bits of m to the result of GenerateWatermark with G

set index i of m to 1

apply bi to o

increment i

algorithm EmbedWatermark

input: Dij, M1..m, G(WG, CG, DG)

output: D'ij

for each markable object (e.g., character) o in Dij

for each watermark method m in M

call MarkObject with m and o along with generator G

return what is now D'ij

Watermark Extraction Algorithm

Let Theories ≈ Interactions = {D1..d1..v •U1..u} be the set of “meetings” between document versions and users, either those recorded as having happened or theoretical possibilities.

Let Marks = {M1..m} be the set of watermarking schemes available.

Let ProbableEmbeddedWatermarks(v, u) = {GenerateWatermark1..50000} be the set of watermarks that were (based on Provenance) or could have been generated in the production of document version Dv by a particular user Uu.

Let ExtractedWatermarkBits(m) = [b1, …, bn]m be the extracted watermark bits for marks Mm recovered from a leaked document.

Let PotentialExtractedWatermarks(m) = {slice(ExtractedWatermarkBits(m), 1, 64), …, slice(ExtractedWatermarkBits(m), n – 64 + 1, 64)} be the collection of 64-bit sequences of ExtractedWatermarkBits(m) beginning at position 1, continuing to position 2, and extending as far along the n bits as possible until they run out. [We actually allow running off either end of the ExtractedWatermarkBits, so the indexes are 1 – 6 and n - 64 + 1 + 6 with the 6 being int(10% \* 64).]

Let Evidence(v, u, m) be the counts of ProbableEmbeddedWatermarks(v, u) that can be found in PotentialExtractedWatermarks(m)

algorithm ExtractWatermark

input: Theories, ProbableEmbeddedWatermarks, PotentialExtractedWatermarks

output: Evidence

for each theory in Theories having a document version v and user u

for each mark in Marks having index m

Evidence(v, u, m) = 0

for each ProbableEmbeddedWatermark(v, u)

if ProbableEmbeddedWatermark is (nearly\*) in PotentialExtractedWatermarks(m)

Evidence(v, u, m)++

return Evidence

algorithm EvaluateEvidence

input: Evidence collected from the ExtractWatermark algorithm

output: U, the user with the most incriminating evidence

for each user U represented in Evidence

let applicableEvidence be all Evidence(v, u, m) where u = U

sums(U) = sum(applicableEvidence)

return U such that sums(U) = max(sums)

\*Nearly means a value exists in PotentialExtractedWatermarks(m) which differs from the ProbableEmbeddedWatermark in at most 10% of the bits.