The watermark embedding “algorithm” is implemented using three different groups of classes in different namespaces and class hierarchies: engine, generator, and scheme. The state variables are split between multiple objects and as a whole this does not convert readily to a representation in pseudo code.

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(WG, CG, DG), the generator, be the tuple of

WatermarkPRNG WG,

ChaffPRNG CG, and

DecisionPRNG DG

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)

WG = new WatermarkPRNG(seed, digest)

CG = new ChaffPRNG()

DG = new DecisionPRNG()

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(WG, 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 watermark 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