Blockchain and Distributed Ledger Technology for Documents

WRITTEN BY JORIS SCHELLEKENS RESEARCH ENGINER AT ITEXT **AND**
WRITTEN BY BRUNO LOWAGIE FOUNDER AND CTO AT ITEXT

CONTENTS

We all know blockchain because it's the technology used by Bitcoin in the context of cryptocurrency. Digital currency is only one application of Distributed Ledger Technology (DLT). In this Refcard, we're going discover when and how to use DLT and blockchain in combination with the Portable Document Format (PDF) to write applications that:

- Give us assurance about the integrity and the authenticity of a document.
- Manage different versions of a document.
- Automate document-oriented workflows.
- And much more.

But first, let's introduce some basic concepts.

BASIC BLOCKCHAIN CONCEPTS

Distributed Ledger Technology is a type of distributed database technology with the following characteristics:

- The records can be replicated over multiple nodes in a network (decentralized environment).
- New records can be added by each node, upon consensus reached by other nodes (ranging from one specific authoritative node to potentially every node).
- Existing records can be validated for integrity, authenticity, and non-repudiation.
- Existing records can't be removed, nor can their order be changed.
- The different nodes can act as independent participants that don't necessarily need to trust each other.

Blockchain is a type of DLT in which records are organized in blocks that are appended to a single chain using cryptography and distributed consensus. Each block contains a timestamp and a link to the previous block. This ensures that data in any given block can't be altered retroactively without the alteration of all subsequent blocks. This approach makes blockchain technology a good choice for the recording of events, records management, provenance tracking, and document life-cycle management.

Some platforms, such as Ethereum, allow developers to build and use decentralized applications running on blockchain technology. This is done using smart contracts.

SMART CONTRACTS VERSUS PDF DOCUMENTS

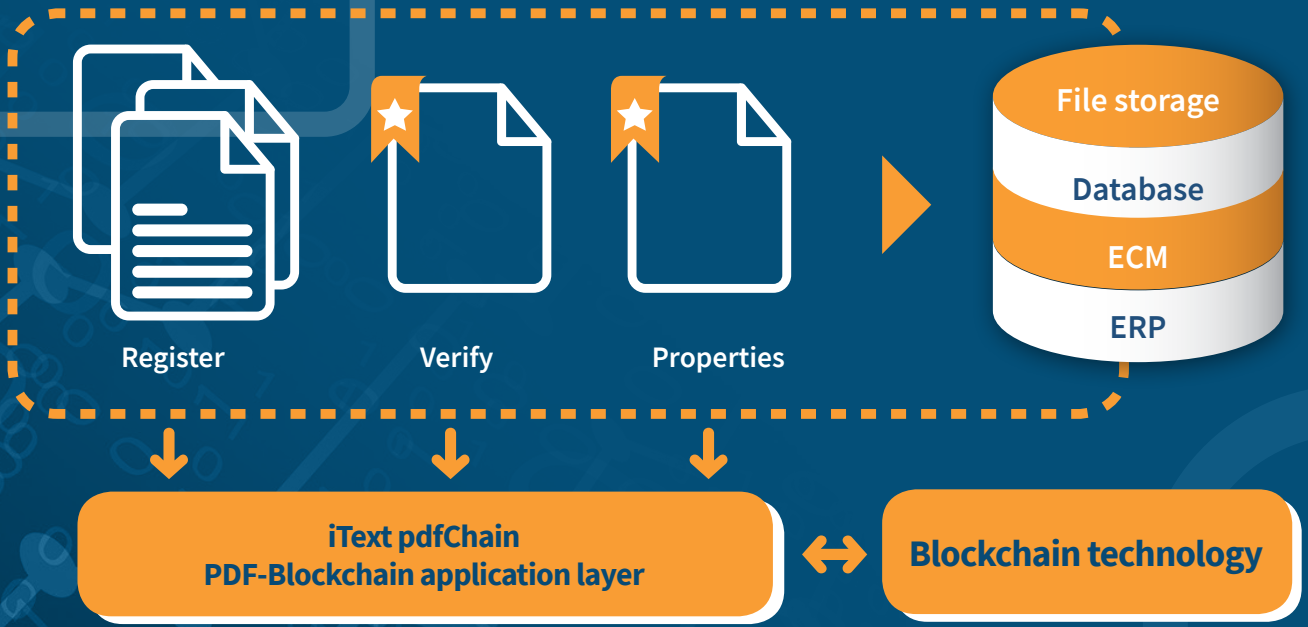
One could build an auction platform for art objects of which the terms and agreements are expressed in source code. For instance:

- A down payment of 50% of the total cost is required before a work of art is shipped.

iText

Blockchain

the future of document security



- ✔ Non-sequential signing
- ✔ Decentralized database
- ✔ Secure certification
- ✔ Software agnostic

Blockchain offers the opportunity to improve document security. This can benefit a wide variety of markets including business, shipping, manufacturing and more by decentralizing the authority authenticating documents. iText is working on an open source version for PDF documents.

Learn more at :
itextpdf.com/blockchain

- Upon delivery of the work of art, full payment of the remaining 50% is automatically processed.

The blockchain will receive events, such as a record that shows a down payment or a record that notifies a delivery. These events trigger the execution of the smart contract, resulting in a shipping order or the processing of a payment. Settlement can happen off-chain or on-chain. We talk about off-chain assets in the case of value-bearing items residing outside of the blockchain; for instance, a physical object such as a painting or a statue. This object usually has a digital counterpart in the blockchain. An RFID tag could be attached to each work of art with an ID that refers to a blockchain record. This allows the object's lifecycle events to be mirrored in the blockchain. For instance, when a work of art changes hands, the blockchain could keep track of its rights and document the owner's claim to it. We refer to on-chain assets when the assets are actually on the blockchain; for instance, the payment is done using Bitcoin.

There are many use cases for this technology, but there are also limitations:

- Not every agreement can be captured in code. For instance, you can't program a marriage certificate.
- Privacy can be an issue:
  - In a public blockchain, aliases are used and info can be encrypted, but that's not 100% waterproof. All data stored in a public blockchain is eventually public.
  - In a private blockchain, there's more security, but not all participants in the blockchain are necessarily trusted. There may be use cases where even a private blockchain doesn't provide sufficient protection.
- Blockchain isn't meant to replace other types of databases and storage. You typically won't store a document in the blockchain because of its potentially large file size and its potential confidential nature.

In this Refcard, we're going to discuss a solution where the off-chain asset isn't a physical object, but a document stored outside of the blockchain as a PDF file.

Since ISO 32000-2 (the PDF 2.0 specification), it is mandatory for every PDF file to have a unique ID. This ID consists of two parts. When a PDF is created from scratch, the two parts are identical. When the document changes, the first part is permanent, whereas the second part changes to ensure the uniqueness of the ID pair. This concept allows different documents with different ID pairs that have the first part of the ID in common to be identified as documents that are related to each other.

We'll use the ID pair of a PDF file, the off-chain asset, as the primary key for the on-chain record in which we'll store a signed hash of the document, along with some metadata. By doing so, we can solve a series of common problems with documents.

AN OVERVIEW OF COMMON PROBLEMS WITH DOCUMENTS

In times of fake news and alternative facts, we want the documents we consume to be genuine. Up until now, the industry has tried to make documents trustworthy by applying a digital signature. Such a signature is created by encrypting a hash of the bytes of the document using a private key. The document is distributed along with this signature and the public certificate of the signer. Whoever receives the document can create a new hash of the document using the same cryptographic hash function, resulting in hash1. The public certificate of the signer consists of the identity of the signer and his public key. That public key can be used to decrypt the signed hash, resulting in hash2. This mechanism guarantees us three things:

1. **Integrity**: If hash1 equals hash2, we know that the document hasn't been tampered with.
2. **Authenticity**: We know who signed the document since this information is present in the public certificate of the signer.
3. **Non-repudiation**: The signer can't deny that they signed the document because they're the only one who is supposed to have access to his private key.

If we also want guarantees about the time of signing, we need to involve a timestamping authority; if we want the signature to be valid on the long term, we need extra Long-Term Validation (LTV) functionality.

These mechanisms have several disadvantages:

- If you want people to trust the signatures in your document, you need to involve some central authorities, such as CAs and TSAs. It would be great if we could think of a system that doesn't require any central authorities.
- Documents that need to be signed by different people can only be signed sequentially. Think of a conference call that requires a dozen people to sign an NDA before the call can take place. Person A receives the NDA document, signs it, and sends it to Person B. Person B receives it, signs it, and sends it to Person C. And so on. Now, suppose that Person D is on vacation the week before the conference call. In that case, all the people who come after Person D can't sign until Person D returns. There's a chance that Persons E, F, G, H, ... won't be able to add their signature in time for the call.

- As documents are signed sequentially, Person A will own a copy with a single signature, Person B will own a copy with two signatures, and so on. If 12 people have to sign, Person L will own a copy of the document with 12 signatures. It is then the responsibility of Person L to send the document that contains all the signatures to all the previous signers. Unfortunately, this doesn't always happen. Often, people early in the chain only have copies that are missing one or more signatures.

- Sometimes, it's hard to know if you're signing the correct version of a document. Think of an agreement that was drafted in a process that involved tough negotiations. During the negotiations, different versions of the agreement were sent back and forth. There's a risk that eventually, the wrong version of an agreement is signed due to a confusingly high number of versions created by different people introducing confusing version numbers. LTV is a cumbersome process that relies heavily on central authorities. Suppose that you've kept a signature alive for a century and you discover that you can't extend the life of the signature because of a sudden, perpetual unavailability of one of the central authorities that were involved. That's problematic.

For all of these reasons, the adoption of digital signatures in PDF has been slow. The functionality was too complex.

When PDF documents aren't signed, you never know if they can be trusted. PDF documents are consumed offline, so there is no green checkmark or padlock informing you that the document was served using the HTTPS protocol, nor any indication whether you can trust the author of the document and the domain from which it was obtained.

Furthermore, if you don't have a copy of a document, but you do have its original URL, you can encounter the problem that the link doesn't work anymore because the document was moved. We call this link rot, and there are centralized services such as doi.org that provide a Digital Object Identifier (DOI) that allows you to retrieve one or more currently active URLs to a resource based on the resource's ID. DOI allows you to register and maintain the location of a document for a fee. Wouldn't it be great if we could reduce that fee and decentralize the service?

We can solve all of these problems, and probably even some problems we didn't even know we had, by introducing blockchain into the world of PDF documents.

BLOCKCHAIN AND DIGITAL SIGNATURES

Suppose that we don't store the signature of a document in the document itself but instead store it in the blockchain. In a blockchain system that relies on PKI, users already have a private key, which already lowers the threshold for adoption. Users of the system could be human beings, companies, or systems used within companies (such as a CRM, ERP, DMS, etc.). In the context of blockchain, the keypair is usually generated in a Web of Trust (WoT), but that's not a must. In some use cases (e.g. in the context of a permissionless blockchain), we might prefer to work with a keypair generated by a CA. As for the private key, some use cases might require that it's stored on a hardware device such as an HSM, smart card, or USB token.

Signing a document would involve creating a hash of the document, signing that hash with a private key, and storing that signed hash in the blockchain along with the corresponding public certificate. In this case, we use blockchain as a database for storing signatures with the unique ID of the PDF as the primary key for each record. Optionally, we can add metadata to the signature records such as the status of the document (draft, final, unpaid, paid, etc.) and one or more locations (e.g. the URLs of different mirrors from which the document can be downloaded).

As a developer, you get a lot of advantages by using this approach. You can be less of a PDF expert: you don't need to worry about incremental updates, which changes are or aren't allowed to the document, and so on. On the blockchain side, you have control over the application (e.g. which metadata to store and which blockchain technology to use), but by using the API explained in this Refcard, all the gritty details about how to query the blockchain will be abstracted away. Furthermore, you get automatic distribution of the signatures and metadata, load balancing, scalability, and security without having to write any code specifically for that purpose.

The address of the relevant blockchain could be stored in the metadata of the PDF. You can use this address to retrieve the record(s) that correspond with the ID pair of the document you want to verify. That record will give you information on the signer or signers. With the hash, you'll be able to verify the document's integrity. If more than one record is retrieved, you'll have a historical overview of who registered the document when, and you can inspect the metadata that was added with each registration.

WHAT ARE THE ADVANTAGES OF USING BLOCKCHAIN FOR DOCUMENTS?

Storing the signature in the blockchain instead of the document itself has many advantages:

- We still meet the requirement of integrity since we create the signature in exactly the same way. We only store the signature in a different place.

- We can still work with a CA to meet the requirements of authenticity and non-repudation, but we don't have to. We can also rely on a Web of Trust stored in the blockchain.

- We don't need to work with a TSA anymore; timestamping is inherent to blockchain. Once a record is added to a block, its contents are immutable and can't be changed.

- Since the signatures are stored outside of the document, signatures can be applied in parallel without a predefined order. Every signer can add a signature to the blockchain at any moment in time.
- Everyone can check who signed a document, when, and in which order; this can be done without having to rely on a central service (e.g. Docusign).
- Keeping the signature of a document alive doesn't require any other approach; any eligible signer with access to the document can create a new signature in the blockchain using the latest hashing and encryption algorithms.

When discussing different use cases, you'll notice that we'll often talk about *registering* a document instead of *signing* a document. Sometimes, we don't want the document to have a legally binding signature. For instance, you might want to register an intermediary draft of an agreement that is being negotiated in the blockchain, but you don't want that contract to be legally binding until it has been finalized and approved by all parties. In some sectors, people won't accept the fact that we store a signed hash in the blockchain as a valid signature, e.g. because the government doesn't accept such a signature as legally binding.

Even in those cases, it makes sense to store hashes of documents along with some metadata in the blockchain.

- You can retrieve the records of a document based on its ID pair to inspect the metadata added by the person who registered the document. For instance, a document that is registered as an invoice with status "unpaid" by the CRM system of a vendor can be registered as an invoice with status "accepted" by the ERP system of a buyer and eventually be registered as an invoice with status "paid" by the buyer's bank.
- You could also do a search on the first part of the ID pair to find records of documents that are related. A document that starts its lifecycle as a quote request could change into a quote, then be accepted as a purchase order, then forked into an invoice and a delivery note.
- If you find a document in the wild, e.g. a draft of a technical specification, and you want to know if that document is the latest version, you could search for a record that registers a more recent version of that draft and use the location information stored in that record to retrieve that updated version.

With the open-source **iText 7 pdfChain add-on**, creating a blockchain record containing a signature of a PDF file is pretty straightforward.

WRITING YOUR OWN IMPLEMENTATION

The pdfChain add-on is blockchain technology-agnostic. You need to implement the IBlockChain interface or pick an existing implementation to match with the blockchain technology of your choice. You also have to extend the abstract class named AbstractExternalSignature.

THE IBLOCKCHAIN INTERFACE

We have defined an interface that consists of only three methods:

- `public boolean put(String key, Record data);`
  A method to put a record on the blockchain.
- `public List<Record> get(String key);`
  A method to get a record from the blockchain based on its key.
- `public List<Record> all();`
  A method to list all records registered on the blockchain.

The Record class simply extends `Map<String, Object>` and is used as a collection of key-value pairs.

THE ABSTRACTEXTERNALSIGNATURE CLASS

We have created an `AbstractExternalSignature` class implementing a `hash()` and an `encryptHash()` method, but four methods are abstract:

- `public abstract String getHashAlgorithm();`
  A method in which you provide the hash algorithm to be used.
- `public abstract String getEncryptionAlgorithm()`
  A method in which you provide the encryption algorithm to be used.
- `public abstract Key getPublicKey();`
  A method in which you provide the public key object.
- `public abstract Key getPrivateKey();`
  A method in which you provide the private key object.

The `DefaultExternalSignature` class currently uses SHA-256 as its hashing algorithm and RSA as its encryption algorithm.

THE PDFCHAIN CLASS

Using an IBlockChain implementation and an external signature class, we can create a PdfChain object that can be used to add and retrieve records from the blockchain.

This is how we add a record to a blockchain for which we used MultiChain technology:

```
IBlockChain mc = new MultiChain(
    "<http://127.0.0.1>", 4352, "chain1",
    "stream1", multichainrpc",
    "BHcXLKwR218R883P6pjiWdBffdMx398im4R8BEwfAxMm");
sign.AbstractExternalSignature sgn =
    new sign.DefaultExternalSignature(
    new File("path_to_keystore"), demo", "password");
pdfchain.PdfChain blockchain =
    new pdfchain.PdfChain(mc,sgn);
File inputFile = new File("input.pdf");
blockchain.put(inputFile);
```

This is how we retrieve records from the blockchain:

```
IBlockChain mc = new MultiChain(
    "<http://127.0.0.1>", 4352, "chain1",
    "stream1", multichainrpc",
    "BHcXLKwR218R883P6pjiWdBffdMx398im4R8BEwfAxMm");
sign.AbstractExternalSignature sgn =
    new sign.DefaultExternalSignature(
    new File("path_to_keystore"), demo", "password");
pdfchain.PdfChain blockchain =
    new pdfchain.PdfChain(mc, sgn);
File inputFile = new File("input.pdf");
for (Map<String, Object> docEntry :
    blockchain.get(inputFile)) {

    for (Map.Entry<String, Object> entry :
        docEntry.entrySet())
        System.out.println(
          padRight(entry.getKey(), 32) +
          " : " + entry.getValue());
    System.out.println("");
}
```

The `get()` method returns List of Record objects because the same document can be stored more than once with different properties.

A possible output could be:

```
blocktime       : 1499691151

id2             : ï¿½Æ±ï¿½ï¿½Bï¿½}Ó™`ï¿½-oï¿½R

id1             : zï¿½L{ï¿½Wd=ï¿½ï¿½ ï¿½ï¿½Gï¿½

publishers      : [14pwDpkcfRvSiw6DJWpP7RdcYgv5NfRRn6Dudr]

txid            : 0092d7eb967ac2e45671742ddf1a0a96bc049a4
                  bbfe3528888b6d9ff396b7a2

hsh             : ï¿½ï¿½Bï¿½ï¿½ï¿½ï¿½ï¿½È™oï¿½$'ï¿½Aï¿½dï¿
                  ½ï¿½Lï¿½ï¿½ï¿½xRï¿½U

confirmations   : 22

key             : ¯Â¿Â½Ã¯Â¿Â½BÃ¯Â¿Â½Ã¯Â¿Â½Ã¯Â¿Â½Ã¯Â¿Â½Ã¯Â¿
                  Â½Ãˆoã¯Â¿Â½$'Ã¯Â¿Â½AÃ¯Â¿Â½dÃ¯Â¿Â½Ã¯Â¿Â½
                  LÃ¯Â¿Â½Ã¯Â¿Â½Ã¯Â¿Â½xRÃ¯Â¿Â½U

shsh            : <garbled>
```

As you can see, the PdfChain class stored an entry with a key named confirmations. That's extra metadata that we stored in the blockchain using the PdfChain method. We can use the BQL functionality of the PdfChain add-on to create a query. For instance, we can use the BQLCompiler to get the ID pair, number of confirmations, and hash value of all the records of which the number of confirmations is greater than 10 but lower than 50.

```
 IBlockChain mc = new MultiChain(
        "<http://127.0.0.1>", 4352, "chain1",
        "stream1", "multichainrpc",
        "BHcXLKwR218R883P6pjiWdBffdMx398im4R8BEwfAxMm");
AbstractBQLOperator op = BQLCompiler.compile(
        "SELECT[id1, id2, confirmations,hsh]"+
        "( confirmations > 10 AND confirmations <50 )"+
        "SORT confirmations");

Executor exe = new Executor(mc);
Collection<Record> resultSet = exe.execute(op);
```

For more examples and more info about the classes used in these examples, see **itextpdf.com/Blockchain**.

USE CASES

USE CASE 1: INVOICES

Imagine that every invoice that is issued is also registered in the blockchain so that the following information can be retrieved:

- The document ID.
- The public certificate of the vendor.
- The hash of the document signed by the vendor.

There's no need to store the amount to be paid, the identity of the buyer, nor any other invoice detail, as that could be a breach of privacy.

There are huge benefits for the customers when the invoice he receives is registered in the blockchain:

- If the invoice isn't registered in the blockchain, it could be a fake invoice. The customer could decide not to pay the invoice until it's officially registered.
- If the hash is signed using a key issued by an official instance, the customer can check if the vendor is who they claim to be and if that vendor has the appropriate licenses to sell goods or services.
- A vendor can't backdate invoices anymore, as there will always be a timestamp of the registration in the blockchain.
- A customer will immediately see if an invoice was forged; for instance, if the invoice was intercepted and the wiring information was changed.

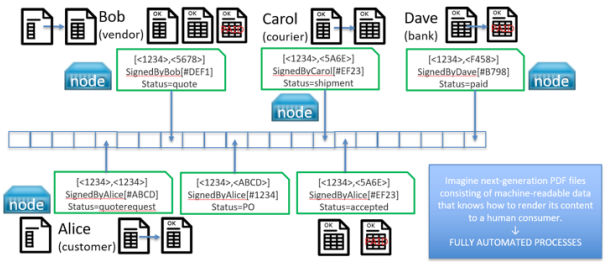
The government, or rather the IRS, will also benefit from this mechanism:

- The IRS could check how many invoices were issued by a company. If a company declares only 800 invoices but the IRS finds 1,000 invoice registrations in the blockchain, there's a problem.
- The IRS could check if all paid invoices paid for by a company were actually registered in the blockchain. If a company declares 1,000 paid invoices but 200 aren't accounted for in the blockchain, there's a problem.

- Not all invoices have to be verified for integrity; a sample check might be sufficient for tax purposes.

USE CASE 2: SALES WORKFLOW

Invoices are merely a first step. We could use the principle for every document in the sales workflow. A document that starts off as a quote request can spawn different quotes being issued by different vendors. After examining the different quotes, one quote could be chosen and result in a purchase order. That purchase order can then fork into an invoice and a delivery note. A bank that processes the invoice can mark the invoice as paid.



The different nodes in this simplified figure represent CRM and ERP systems at different companies. Each time an invoice leaves the CRM at the vendor, that invoice could be registered automatically in the blockchain. When this invoice is received in the ERP system on the buyer's side, it can be validated in the blockchain before it is sent to the bank to process the payment. Once the bank marks an invoice as paid, the CRM of the vendor will notice this immediately when the block containing a record created by the bank reaches the CRM's blockchain client.

SUMMARY

Signing a PDF document in the blockchain instead of storing a signature in a PDF reduces the complexity of the code for a developer who needs digital signing and verification functionaliy. The same principle can be used in many other use cases to implement a document workflow, keep track of the location of a document, and much more. Software integrators can introduce the pdfChain add-on into Document Management Systems (DMS), Customer Relationship Management (CRM), and Enterprise Resources Planning (ERP) systems. They can add the pdfChain add-on on top of storage solutions and secure document vaults. The blockchain will help them keep track of those documents and inherently provide digital signature functionality.

Written by Bruno Lowagie

Bruno Lowagie is the original developer of iText, an innovative PDF library that has grown into a global software company. As an active member of the ISO and PDF communities he has authored several books about iText as well as worked with the community to continually improve PDF functionality. When he is not working in the PDF world, Bruno spends much of his time with his wife and two sons.

Written by Joris Schellekens

Joris Schellekens is a Research Engineer with iText who focuses on disruptive technologies such as: neural networks, machine learning and natural language processing. He is passionate about new technology and finding ways to solve challenges with PDF. When he is not researching or coding for iText, you can find him working on his own coding projects, working out complex mathematics equations, or listening to music.

DZone, Inc.
150 Preston Executive Dr. Cary, NC 27513
888.678.0399 919.678.0300