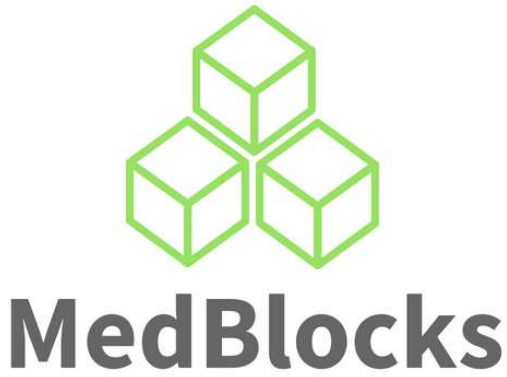
Sidharth Ramesh
Interested in: applying deep learning, blockchain, making videos, music, write stories, taking photos, kaggling.
May 26 · 13 min read

Introducing MedBlocks —Storing Medical Records Securely on the Interplanetary File System using Blockchain technology



There is a video demonstration at the end. Skip to that if you are impatient.

A very obvious problem that plagues the medical field is the nature in which medical records are stored. Let's forget about the non-electronic records for a second, and just concentrate on the Electronic Health Record (EHR) for now.

Every hospital that you visit has its own Record Management Software. Some store it locally in their databases, others use a cloud service provider. Some store the data in a format compliant with insurance agencies, others just don't care. Most of the time, your data is on a server that belongs to the hospital or is rented by the hospital.

The major problems that this model causes

1. Fragmentation of your medical information across hospitals, private medical practitioners, and m-health apps.

2. Inability to transfer your records from one hospital or application to another.
3. Inability to access vital medical information in case of emergencies.
4. Data leaks and hospitals selling your information to companies that benefit from it.
5. Manipulation of data by hospital authorities
6. Unauthorized access to private medical data

Medical data is valuable and needs to be protected

There is no doubt that data is the oil of the future economy. With machine learning algorithms becoming more and more robust, big companies will need more and more data. Social media platforms like Facebook are already facing the backlash of selling your data. Data is valuable intrinsically, and in the future, companies and big corporate may even pay you for your data. This may lead to a Universal Basic Income, in which you get money for just generating data, but that's a story for another day.

The take-home message is this: **Guard your data like you would guard you other valuables like gold, money, or cryptocurrencies. Because your data is valuable in the exact same way. *Your medical data information, even more so.***

I started working on MedBlocks inspired by another project in Manipal University that stores medical information on RFID cards.

MedBlocks makes it possible for anyone to add medical information about you on a public database. But don't worry, no one will be able to make sense out of anything until you give them permission explicitly. All records created will be stored on a public blockchain, and cannot be manipulated at will. It tries to solve the problems mentioned above.

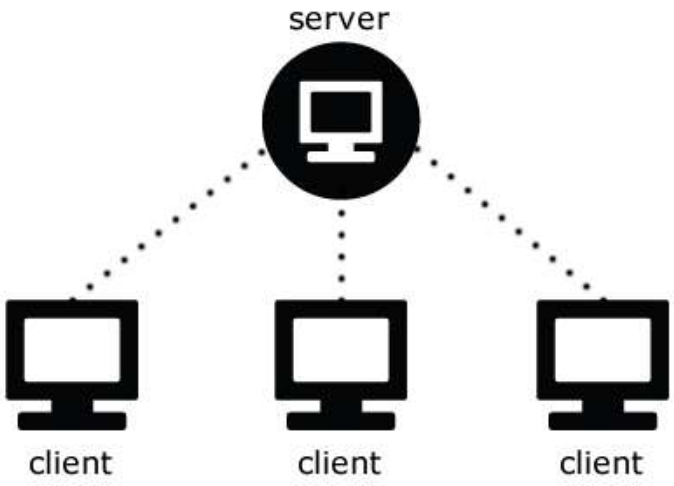
The rest of the article will describe the working of MedBlocks and the technologies that make it possible.

Almost all of the technologies have developed rapidly in the past decade, so I hope it serves as a good application based study on some bleeding edge technology.

IPFS—The Interplanetary File System

We all know and love HTTP, the protocol that makes it possible for you to view this page right now.

The HTTP protocol was based on a client-server model, in which any content you want to retrieve resides in a server, and you, the client asks for the content by first contacting the server. The server then responds by serving you with the content that you requested.

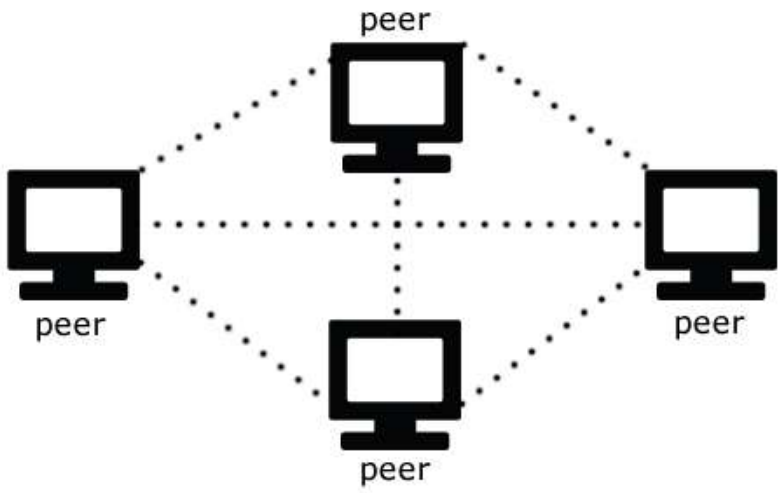


The client-server model

This obviously has its downsides. One of the major ones being, fragmentation of data across multiple servers in our case of medical records. Each hospital has the data on its own server, and to retrieve that data, you first need to contact the server. Sometimes the server may be even private or inaccessible.

Enter IPFS!

I highly recommend at least just scrolling through their website at ipfs.io, because it's just beautiful. The main concept is, instead of addressing content by the server it is stored on, address it by the content itself, (or more specifically it's hash). And this content will be retrieved from nearest computer.



This is just such a brilliant idea that I keep thinking why didn't we think of this before! It's kind of like how torrents work, but even better. If you want a movie and your roommate has a copy of it, when you request the movie, by its hash, it will retrieve it from your roommate through the local network at full WiFi speed! And this without you or your roommate knowing that this transfer happened! How cool is that?

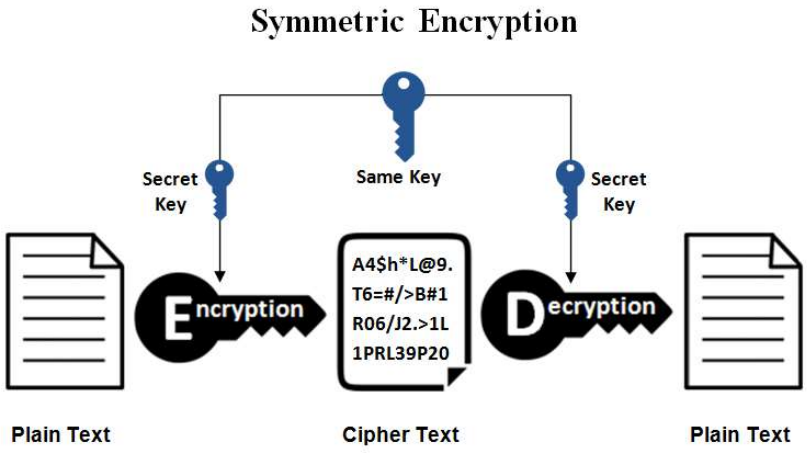
MedBlocks uses IPFS to store data. Meaning, if you just stay in one hospital the records stay there, and if you move to some other hospital and request for your records, it will just follow you. As long as someone with your data is running an IPFS node, your data can be accessed from anywhere at all times!

That brings us to our next concern…"If it's online at all times, then anyone can access it right?"—Privacy and Security of your medical records.

Symmetric and Asymmetric Encryption

To make our records private such that only the ones that we authorize have access to it, we will need to use some form of encryption. It's essential here to understand the basics of cryptography some basic forms of encryption.

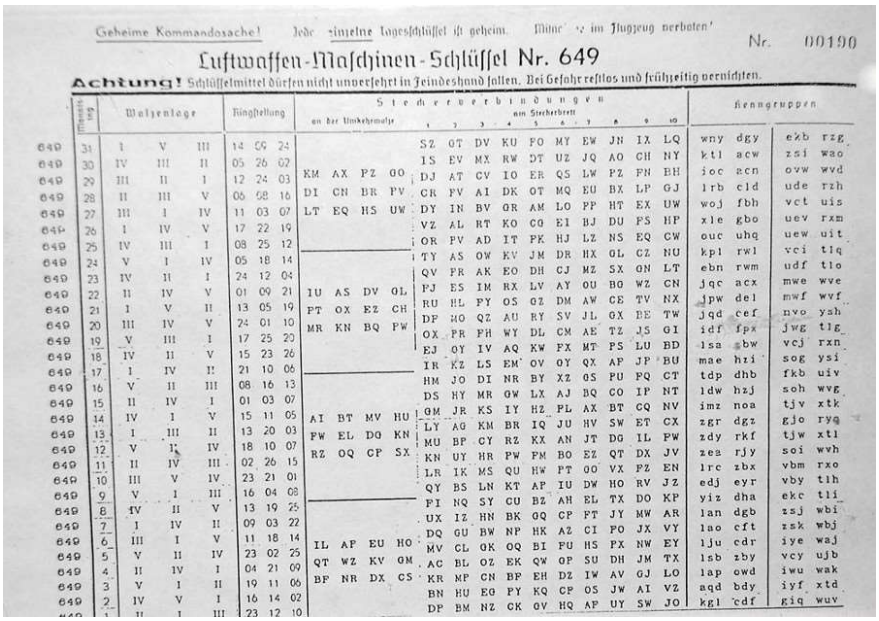
First, we have symmetric encryption



It's pretty straightforward. You have a key (also called a cipher) that "encrypts" a document into a jumbled mess. To retrieve the original

document you need to “decrypt” it with the same key. A popular symmetric algorithm today is AES.

This method of encryption was used extensively, even during the cold wars. The problem was, the key had to be exchanged securely in person. This proved to be more difficult than imagined, since enemy spies, soon figured out the key from the pieces of papers that were being passed on. Somehow :-(


Geheime Kommandosache! Jede einzelne Tagesschlüssel ist geheim. Mitnahme im Flugzeug verboten!

Nr. 00190

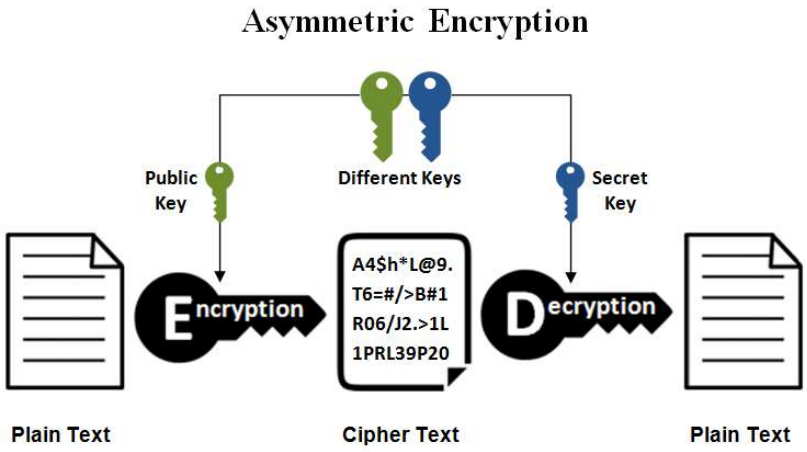
Luftwaffen-Maschinen-Schlüssel Nr. 649

Achtung! Schlüsselmittel dürfen nicht unversehrt in Feindeshand fallen. Bei Gefahr restlos und frühzeitig vernichten.

Monat	Tag	Walzenlage	Ringstellung	Steuerverbindungen an der Umkehrwalze	Steuerverbindungen am Steckerbrett 1 2 3 4 5 6 7 8 9 10	Kenngruppen
649	31	I V III	14 09 24	KM AX PZ GO	SZ GT DV KU FO MY EW JN IX LQ	wny dgy ekb rzg
649	30	IV III II	05 26 02	DI CN BR FV	IS EV MX RW DT UZ JQ AO CH NY	ktl acw zsi wao
649	29	III II I	12 24 03	LT EQ HS UW	DJ AT CV IO ER QS LW PZ FN BH	ioc zcn ovw wvd
649	28	II III V	06 08 16		CR FV AI DK OT MQ EU BX LP GJ	lrb cld ude rzh
649	27	III I IV	11 03 07		DY IN BV GR AM LO PP HT EX UW	woj fbh vct uis
649	26	I IV V	17 22 19		VZ AL RT KO CG EI BJ DU FS HP	xle gbo uev rxm
649	25	IV III I	08 25 12		OR FV AD IT PK HJ LZ NS EQ CW	ouc uhq uew uit
649	24	V I IV	05 18 14		TY AS OW KV JM DR HX QL CZ NU	kpl rwl vci tlq
649	23	IV II I	24 12 04	IU AS DV GL	QV FR AK EO DH GJ MZ SX GN LT	ebn rwm udf tlo
649	22	II IV V	01 09 21	PT OX EZ CH	FJ ES IM RX LV AY OU BQ WZ CN	jqc acx mwe wve
649	21	I V II	13 05 19	MR KN BQ FW	RU HL PY OS GZ DM AW GE TV NX	jpw del mwf wvf
649	20	III IV V	24 01 10		DP HQ QZ AU RY SV JL QX BE TW	jqd cef nvo ysh
649	19	V III I	17 25 20		OX FR FH WY DL CM AE TZ JS GI	idf fpx jwg tlg
649	18	IV II V	15 23 26		EJ QY IV AQ KW FX MT PS LU BD	lsa sbw vcj rxn
649	17	I IV II	21 10 06		IR KZ LS EM OV OY QX AP JP BU	mae hzi sog ysi
649	16	V II III	08 16 13		HM JO DI NR BY XZ OS FU FQ CT	tdp dhb fkb uiv
649	15	II IV I	01 03 07	AI BT MV HU	DS HY MR GW LX AJ BQ CO IP NT	ldw hzj soh wvg
649	14	IV I V	15 11 05	FW EL DQ KN	GM JR KS IY HZ PL AX BT CQ NV	imz noa tjv xtk
649	13	I III II	13 20 03	RZ OQ CP SX	LY AG KM BR IQ JU NV SW ET CX	zgr dgz gjo ryq
649	12	V II IV	18 10 07		MU BP CY RZ KX AN JT DG IL PW	zdy rkf tjw xtl
649	11	II IV III	02 26 15		KN UY HR PW FM BO EZ QT DX JV	zea rjy soi wvh
649	10	III V IV	23 21 01		LR IK MS QU HW PT OO VX PZ EN	lrc zbx vbm rxo
649	9	V I III	16 04 08		QY BS LN KT AP IU DW HO RV JZ	edj eyr vby tih
649	8	IV II V	13 19 25	IL AP EU HO	FI NQ SY CU BZ AH EL TX DO KP	yiz dha ekc tli
649	7	I IV II	09 03 22	QT WZ KV GM	UX IZ HN BK GQ CP PT JY MW AR	lan dgb zsj wbi
649	6	III I V	11 18 14	BF NR DX CS	DQ GU BW NP HK AZ CI PO JX VY	lao cft zsk wbj
649	5	V II IV	23 02 25		MV CL GK OQ BI FU HS PX NW EY	lju cdr iye waj
649	4	II IV I	04 21 09		AC BL OZ EK QW GP SU DH JM TX	lsb zby vcy ujb
649	3	V I II	19 11 06		KR MP CN BP EH DZ IW AV GJ LO	lap owd iwu wak
649	2	IV V I	16 14 02		BN HU EQ FY KQ CP OS JW AI VZ	aqd bdy iyf xtd
649	1	II I III	23 12 10		DP BM NZ GK OV HQ AP UY SW JO	kgl cdf giq wuv

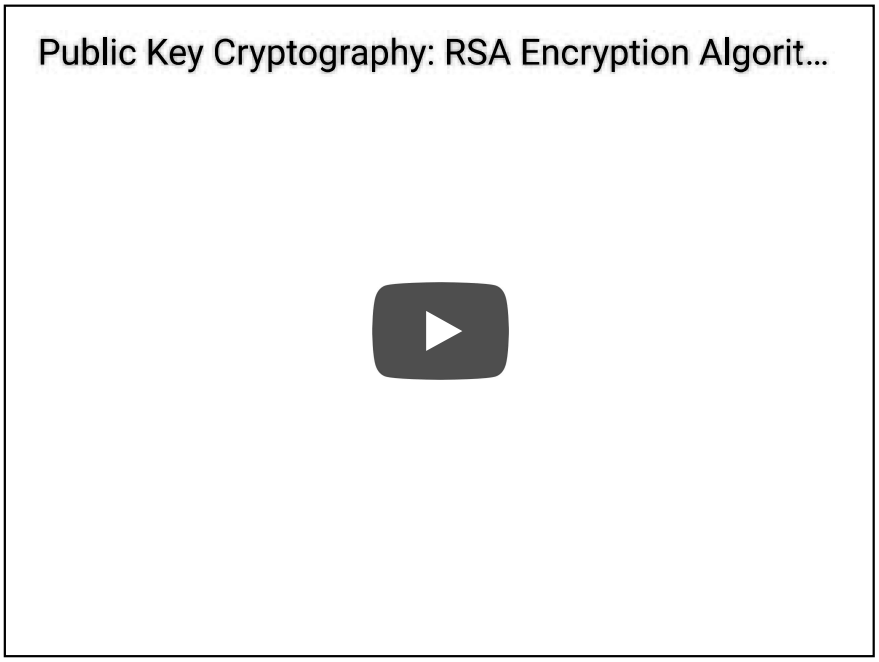
Keys to the German Enigma Cipher Machine during World War 2

Then came Asymmetric Encryption (also known as Public key cryptography), just about 60 years ago. This is the most pirated piece of software to ever exist and for a good reason.



In this method, you have two keys. A public key, and a secret key (also called a private key). To encrypt a message you use the public key. However, to decrypt it, you need the secret key. This means even the person who encrypted the message cannot decrypt it. RSA is a very popular Asymmetric encryption algorithm.

The public key and the secret key are related mathematically, but the secret key cannot be derived from the public key. The whole method has a lot to do with prime factorization, modulus operations and a lot of other nerdy stuff. If you want to really understand, I recommend this video:



A Public Immutable Database

That's probably the most underwhelming title given to Distributed Ledger Technology, but for our use case, it's just that. A public database that everyone can trust, because everyone or at least a lot of people have a copy of it.

We'll be using this to store important information regarding our data, patients and doctors, but not the data itself. **We'll be using IPFS for that.**

I've looked at options like BigChainDB, QTUM, Quorum, Hyperledger. Working on Ethereum using Events as cheap storage and using BigChainDB's were the most appealing options.

Option 1: Ethereum

To store data on the Ethereum main network, one has to pay a fee. This will be a huge turn off for many users. The Development networks or the Test networks aren't equally trustworthy but do not need any transaction fees, because the ether can be obtained for free. A network of hospitals can run their own private version of the ethereum test net, and therefore will technically be free, although they have to maintain private ethereum nodes.

The reason for storing this on the blockchain in the first place is to ensure that the records are not altered. A permanent reference must exist of every medical record ever created. For that, we need the network to be as decentralized as possible.

Developer side notes:

Storing data on the Ethereum blockchain in a smart contract is very expensive. As much as $1000 for every 1MB on the main net!!

A smarter approach to store data cheaply would be to use Event Logs. This was meant as a way for smart contracts to interact with the front end, but it also doubles as a cheap storage option since all logs are included with the transaction in the blockchain forever.

A smart contract the just acts as an interface to write events is all we need.

Tk. Insert smart contract image

However, with this approach, I had some trouble retrieving the Events from web3.py due to encoding errors. I skipped this approach to go with bigchainDB instead, out of the mere frustration with Ethereum's/web3.py's errors with encodings.

Option 2: BigChainDB

BigchainDB is kind of like a variation of MongoDB to incorporate blockchain like features.

The main problem with this is that it's currently centralized. But in the future, this may not be the case, with the main net launching.

Developer notes: Assets are used to store the data. These assets can be transferred around. A metadata field can be added with every transaction, and so, data can be appended this way.

For the first working version of medblocks, the bigchainDB test network was used. It is not different from the main network so porting will probably not be a problem.

Let's build MedBlocks!

Now that we have all these technologies, let's come back to the problem at hand. How do we use these cryptography techniques, the blockchain to securely store our medical data on an open database like the IPFS?

I've used a combination of the encryption techniques in a way that made the most sense to me. None of the heavy data is stored on the actual blockchain, but is offloaded onto the IPFS.

Here is how the whole thing works:

Step 1: Declare your identity and a public key

Everyone on the network—the patients, the doctors, the apps first generate an RSA key pair. They keep the secret/private key a secret (duh!) stored safely, while they make the public key, well…public.

So our public record on the blockchain would look something like this:

```
Patient 1: rsa_public_key: jdshfksjudhf8923045u02
Doctor 1: rsa_public_key: skdjhfs98urw9384u5rw324
Lab 1: rsa_public_key: 234sdfsdfsf43wwertewrtwe
```

Each participant will have declared a public key, while having the corresponding private key.

Step 2: Health care providers generate data

This can be anyone who documents some data on the patient—A doctor, a laboratory, a fitness tracking app, or the patient themselves.

For now, we'll assume it to be a Glucose report coming from Lab 1 for Patient 1.

```
Glucose Report
--------------

Date: 23.2.20
Name: Jenny Tallia
Age: 23

Serum glucose - 110 mg/dL
Method: Glucose Oxidase
```

Ok. So, out imaginary Jenny Tallia is the patient and her glucose levels, are borderline normal. That's the data we'll be working from now on.

Step 3: Encrypt the data using a random AES key

We'll be using the AES symmetric encryption algorithm to encrypt this data. We first need a key. We'll generate this randomly.

I'm using python, but as long as you understand, it's not a big deal.

```
>>> import os
>>> os.urandom(32)
b'vq\xa7\xe5\xf640\xfeJ\xbf\xbaT)\x8e\xae\x8b\xa5`5\x8aq_\x0
b\x0bh\x91M\xe9\xaf2\xb7F'
```

Actually, just assume the AES key to be 1234whatever instead of whatever the heck we actually got. Demonstrating code here was probably a bad idea. Check my GitHub repository, at the end of this post for code.

```
Glucose Report
--------------


Date: 23.2.20
Name: Jenny Tallia
Age: 23


Serum glucose - 110 mg/dL
Method: Glucose Oxidase
```

AES encrypt with 1234whatever as key:

```
sdfsdferkjwkerwernfsdfsdf28342io23k4h23kj4
324j23hg4iu23g4riu23krgjk32wegrkj3brkj23r23
'r23urg23iurg3rwebrjkwabrhrkjhkj2h434
4krwjkerkwjaherlawjehrkjwehrkjwaehrkjwehrkjw
'34hjwgerjywag3487i23urrwjkefhkaef,erfwejfbrwekj
hersjfgjsehfgkwejhfjwelhfkjwhrkjwehrjkehrkjwe
```

Now we get the encrypted file. This can only be decrypted with 1234whatever. (Don't even try to actually decrypt that. I just randomly typed)

Step 4: Asymmetrically Encrypt the AES cipher for the patient

So anyone with the encrypted file and 1234whatever can decrypt our file. Lab 1 generated it randomly, but and wants Patient 1 to know his results.

Patient 1's public RSA key has been declared on step 1. So let's get that.

```
Patient 1: rsa_public_key: jdshfksjudhf8923045u02
```

We got our patient's public key. Now to encrypt this:

```
1234whatever
```

Encrypt *1234whatever* with *jdshfksjudhf8923045u02* using RSA.

```
2i834yraekujjksdfhsjkdej
```

That is the encrypted form of 1234whatever. Only Patient 1 can decrypt that since only he has got the private key. And once he has decrypted the key, he can decrypt the contents of the actual report.

Step 5: Store the encrypted data on IPFS and make the encrypted key public

Lab 1 now just needs to put this on the IPFS:

```
sdfsdferkjwkerwernfsdfsdf28342io23k4h23kj4
324j23hg4iu23g4riu23krgjk32wegrkj3brkj23r23
'r23urg23iurg3rwebrjkwabrhrkjhkj2h434
4krwjkerkwjaherlawjehrkjwehrkjwaehrkjwehrkjw
'34hjwgerjywag3487i23urrwjkefhkaef,erfwejfbrwekj
hersjfgjsehfgkwejhfjwelhfkjwhrkjwehrjkehrkjwe
```

Once stored on the IPFS, it generates a hash, which is unique to that record. Anyone with the hash can get the data. Here is the hash:

```
Qxsdfsdkfjlwskejr034034902343
```

And now store publicly on the blockchain the following in what I like to call “a MedBlock”:

```
MedBlock 1 for Patient 1
ipfs_hash: Qxsdfsdkfjlwskejr034034902343
keys:
 - Patient 1: 2i834yraekujjksdfhsjkdej
```

Notice that under the keys, we only have Patient 1 for now.

Step 6: Patient views report

Patient 1 is intimated, since a MedBlock has been added publicly.

```
MedBlock 1 for Patient 1
ipfs_hash: Qxsdfsdkfjlwskejr034034902343
keys:
 - Patient 1: 2i834yraekujjksdfhsjkdej
```

He can easily retrieve *Qxsdfsdkfjlwskejr034034902343* from the IPFS by just requesting for the file by that hash.

Now to view, the patient, first needs to decrypt the AES key meant for him. He does so with his private key:

```
2i834yraekujjksdfhsjkdej
```

Decrypt with rsa_private_key(23ui4yiuerhfskjdfskdfskd) using RSA

```
1234whatever
```

Now the patient has got 1234whatever, which was the random key that was generated in Lab 1.

Now this file retrieved from the IPFS:

```
sdfsdferkjwkerwernfsdfsdf28342io23k4h23kj4
324j23hg4iu23g4riu23krgjk32wegrkj3brkj23r23
'r23urg23iurg3rwebrjkwabrhrkjhkj2h434
4krwjkerkwjaherlawjehrkjwehrkjwaehrkjwehrkjw
'34hjwgerjywag3487i23urrwjkefhkaef,erfwejfbrwekj
hersjfgjsehfgkwejhfjwelhfkjwhrkjwehrjkehrkjwe
```

can be decrypted with 1234whatever using AES to:

```
Glucose Report
--------------


Date: 23.2.20
Name: Jenny Tallia
Age: 23


Serum glucose - 110 mg/dL
Method: Glucose Oxidase
```

👏🎉👏🎉👏🎉

Step 7: Patient gives permission for Doctor to view:

Now our Patient 1 goes to Doctor 1 to get an opinion of the result. Doctor 1 can retrieve all information about the patient by just looking it up on the blockchain. He can retrieve every single report by it's IPFS hash.

But, wait he can't view anything, because it's encrypted by our strong AES algorithm. How will the patient grant the doctor permission to view his report?

The doctor's public key is on the blockchain:

```
Doctor 1: rsa_public_key: skdjhfs98urw9384u5rw324
```

Now the patient just encrypts the AES key using this public key

```
1234whatever
```

RSA encrypt with *skdjhfs98urw9384u5rw324*

```
qw3erwerersedfsdfwer
```

and adds it under the keys for that MedBlock, as a key meant for Doctor 1.

```
MedBlock 1 for Patient 1
ipfs_hash: Qxsdfsdkfjlwskejr034034902343
keys:
 - Patient 1: 2i834yraekujjksdfhsjkdej
 - Doctor 1: qw3erwerersedfsdfwer
```

And done! Doctor 1 and only Doctor 1 can now decrypt the AES key using his private key, and then decrypt the report with that.

Notice the granularity of the permission granted. Doctor 1 has access only to MedBlock 1 that contains his glucose report from Lab 1. Doctor 1 obviously knows that the patient has other medical information, since it's stored on the blockchain, but unless the patient grants permission, but adding a key for the doctor, it remains encrypted.

Step 8: Patient wants to delete a record:

The patient can add a *approved:False* flag to the MedBlock.

```
MedBlock 1 for Patient 1
ipfs_hash: Qxsdfsdkfjlwskejr034034902343
keys:
 - Patient 1: 2i834yraekujjksdfhsjkdej
 - Doctor 1: qw3erwerersedfsdfwer
approved: False
```

It will still remain on the blockchain forever and doctors will still be able to see it encrypted, but it will just indicate that the patient has refused this particular record. A history of everything is always maintained on the blockchain and nothing can really ever be deleted.

Emergency data:

If a piece of information is known to be vital during emergencies, that can be encrypted with 2 keys at the creation of the medblock instead of one. The emergency public key can be declared in the source code, or in some other open way.

```
MedBlock 1 for Patient 1
ipfs_hash: Qxsdfsdkfjlwskejr034034902343
keys:
 - Patient 1: 2i834yraekujjksdfhsjkdej
 - Emergency: 234dsfsfsdfsdfwserfsdf
```

Also to make sure that this emergency address is not misused, an ethereum address with a significant amount, whose private key is encrypted with the emergency public key can be announced.

This way, anyone who has access to the emergency account, is incentivized to protect the private key, because a lot of money is at stake.

That covers the basic functionality of how MedBlocks works.

Demonstration—MedBlocks v0.01

Before starting work on MedBlocks, I surveyed the blockchain space for other similar projects. I found the following big names: Medicalchain, Proof.work, Medichain and a lot more…

All of them have a nice website, a whitepaper on what they would do, and OF COURSE, an ICO Token Sale. I get that people are trying to make some money, but NONE of the projects actually have a working product!! No code, no prototype!!

And then I saw this from Loom network, who have one of the best Solidity tutorials at CryptoZombies.io.

> *"Loom Network doesn't have a whitepaper because Loom Network is too busy shipping code."*

That inspired me a lot. So, I have made a simple command line tool in python that implements MedBlocks. There is still a lot of work to be done, but this is the Minimum Viable Product and it works.

Considerations

Why is the encryption so complicated?, Encrypting the key needed to decrypt…that's such a tongue twister!, Why store the data on IPFS and not on the private blockchain if gas is free?

I reached many of the design decisions after the following considerations:

1. RSA is very slow to encrypt large amounts of data
2. The blockchain is not good to store large amounts of data

3. Granularity of permission must be maintained
4. Anyone/any app must be able to generate and write medical data for the patient without anyone's permission, including the patients'.
5. Patient must be able to control his data, but a history of changes must be maintained

Future direction

Of course, the current demonstration is just a prototype. Most problems concerned with humans aren't the one related to technology. It's with implementation and adoption. To make things easier and actually useful, there is still a lot of work to be done.

1. Storing user data (.json file) on RFID patient cards
2. Creating a JAVA version of MedBlocks to integrate with Android apps
3. Integration with popular EHR management software

I work on this project in my free time. Any help is welcome! If you are a developer and would like to contribute just send me an email at *tornadoalert@gmail.com*.

You can also contribute to the MedBlocks project by making a pull request: https://github.com/sidharthramesh/medblocks. My code is probably a mess right now, but I promise to clean it up if you start contributing. Promise. Even if you can't code, you can help with the documentation.

Thank you for spending time reading this post. Applaud and share if you like this, so it can reach more people.