QRStream: A Secure and Convenient Method for Text Healthcare Data Transferring

Huajian Mao, Chenyang Chi, Jinghui Yu, Peixiang Yang, Cheng Qian and Dongsheng Zhao

Abstract— **With the increasing of health awareness, the users become more and more interested in their daily health information and healthcare activities results from healthcare organizations. They always try to collect them together for better usage. Traditionally, the healthcare data is always delivered by paper format from the healthcare organizations, and it is not easy and convenient for data usage and management. They would have to translate these data on paper to digital version which would probably introduce mistakes into the data. It would be necessary if there is a secure and convenient method for electronic health data transferring between the users and the healthcare organizations. However, for the security and privacy problems, almost no healthcare organization provides a stable and full service for health data delivery. In this paper, we propose a secure and convenient method, QRStream, which splits original health data and loads them onto QR code frame streaming for the data transferring. The results shows that QRStream can transfer text health data smoothly with an acceptable performance, for example, transferring 10K data in 10 seconds.**

I. INTRODUCTION

As the user health awareness increases, the data management need for personal health becomes more and more intense [1] [2]. It becomes common for the users to own [3] [4] and record their daily health information and their activities happened in the health organizations [5] like in the hospitals for different purposes [6].

There are lots of Apps (e.g. Apple Health, Google Health) to collect the daily health data, such as exercise record, heart rate, and blood pressure. However, it is not easy to manage the healthcare organization data. The problem mainly comes from that most of the results are provided in paper but not in digital. If the users want to merge all their health records into some Apps, they have to either input the paper result manually or take picture of the paper result and attach the picture to the App. But these methods may introduce result mistakes or lead to the hardness of the further text health data usage. It would be much better if the healthcare organizations could provide the health record in digital.

Theoretically, the healthcare organizations can authorize the disclosure of the user's health data on the Internet as a service, so that the users can access their health record through the network. However, it would probably bring the risk of data servers being hacked or causing leakage of the private data such as user health, or bringing potential legal issues. Therefore, for the secure and privacy reasons [7], only limit data access methods [8] are provided, and no organization is will to provide the full user health data in the form of centralized services. In addition, the healthcare organizations can technically deliver the users data through connected methods like flash disk, CD, etc, however, this method is usually inefficient and the user operation is extremely cumbersome. At the same time, it makes the organizations in a high risk state of the device being infected with the virus. Therefore, almost no healthcare organization, at least in China, would like to provide users an electronic health record delivery method.

For these reasons, it is urgent and necessary to provide a secure and convenient method for health data transferring between the users and healthcare organizations. In this paper, we focus on the text health data and propose QRStream, a secure and convenient method for text healthcare data transferring. QRStream splits the original data into small slices and encodes them into QR code figures and then plays them on the screen, while in the client side, the client can use camera to record and extract the QR code figures and then merges the extracted data into the original data to finish the data transferring from the server side to the client side.

II. METHODS

We propose QRStream to using QR code streaming to transfer text healthcare data from one device to another. Figure 1 is an overview of QRStream. As the figure shows, QRStream contains two main part, server side Data Encoder and client side Stream Decoder. The Data Encoder is always resident in the server side, for example on the record printing terminal in the hospitals or other servers. Its main duty is to transform the original text healthcare data to the QR code streaming and then play them on the terminal screen for the users who want to take out their own healthcare record in digital. While the Stream Decoder is always a component of the client side App. It is used to capture and decode the QR code streaming and then decode the information and merge them into the original data so as to finish the data transferring from the server side to the user device side.

A. Data Encoder

Data Encoder splits the original data to be equally split into slices and prepends extra header information, including slice number, tag information and so on, to each slice. And after the data spliced, Data Encoder will encode them into QR code frames and play the stream on some terminal screen for the client Apps to scan them. Besides, the Data Encoder should tell the client side App several important parameters,

The authors are with faculties of Academy of Military Medical Sciences, Beijing, China. Corresponding author: Dongsheng Zhao. Emails: {hjmao, chicy, weijh, yangpx, qiancheng, dszhao}@bmi.ac.cn

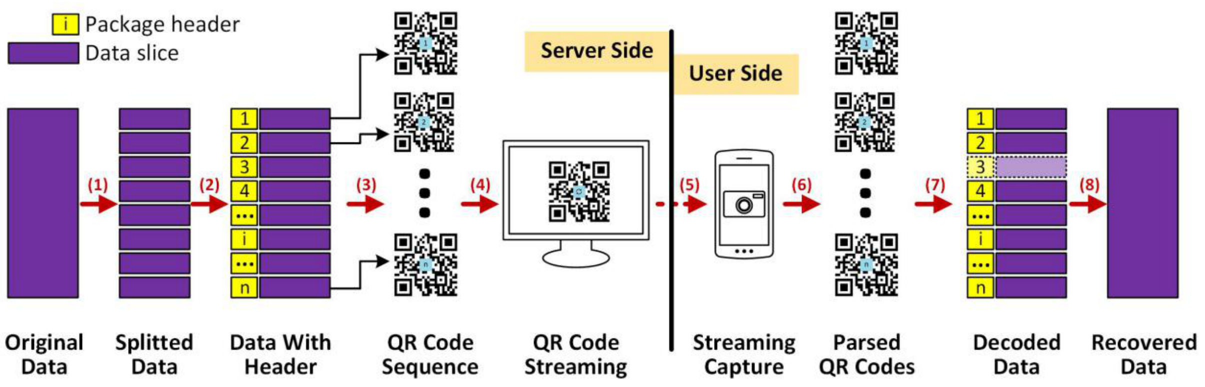

Fig. 1. Workflow overview of QRStream

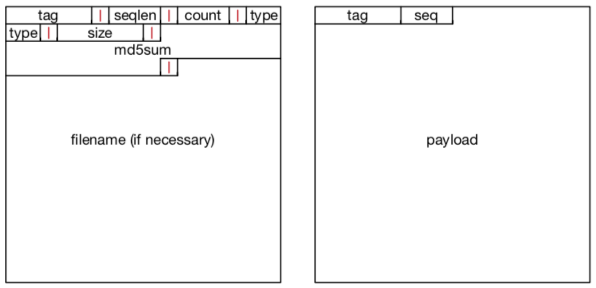

Fig. 2. QRStream protocol package format

including the md5sum of the data for content verifying, a tag of the transferring task for identification, the count of the data slices and so on. Only with these parameters, the client side App can decode the transferring data correctly. For this purpose, a QR code frame of the metadata will be also generated for the parameters negotiation.

Figure 2 shows the payload and metadata QR code frames package format of the QRStream transferring protocol.

In the metadata frame, QRStream contains the parameters including `tag`, `seqlen`, `count`, `size`, `md5sum`, `type`, and `name`. In these parameters, the `tag` is used to uniquely identify the transferring task, and the `seqlen` is the length of slice number in the payload frame package, and the `count` tells the count of the slices of the whole data, and the `size` is the length of the transferred data, and the `md5sum` is the md5sum value of the transferred data, and the `type` is the content type which is `FILE` (which means the content to be transferred is a file) or `TEXT` (which means the content to be transferred is some text content), and `name` is the file name of the transferred data if the `type` is `FILE` or blank if the `type` is `TEXT`. As some of the parameters are variable in length, we separate each parameter with the character `'|'`, and compose the form `'tag|seqlen|count|size|md5sum|type|size'` in text content.

The payload frame includes three information which are the `tag` value, the `seq` number and the raw slice data. The `tag` value is used to check if it is one of the transferred slices by comparing it to the `tag` value in the metadata frame. Only if the `tag` value is valid, the QRStream will add the raw slice data into the receiving buffer array according to the `seq` number for the stream decoding steps.

B. *Stream Decoder*

In order to capture the data from the server side Data Encoder, the Stream Decoder component should be integrated into the client side App.

In the common usage steps, the Stream Decoder will first capture the QR code of the metadata frame, and decode the parameters from the metadata QR code. Then the decoder should be used to capture the played payload data QR code stream, and decode them with the metadata `tag` and `seqlen` values. Ideally, all the payload frames should be captured by the decoder component. However, because of the external environment like the shake of the camera-holding hands or the environment light, some payload frames might be missing in the streaming capturing step. In this situation, the user can tell the Data Encoder the missing frame sequence number, and the QRStream will display the missing frame statically (not in the QR code stream), and waits the decoder to capture the missing frame.

After all the data frames are captured, which means all the slices are received, the data will be merged together according to the sequence number to form the original raw data. And the Stream Decoder will calculate the md5sum value with the merged raw data, then compares it with the `md5sum` value in the metadata frame. If the two values are equal, then the data is transferred successfully, and the client side App can use the merged data to save to a file or to display the content. Or if the two values are not equal, which means some slices are decoded incorrectly, the user should restart all the steps from capturing the metadata. However, benefit from the QR code correction mechanism [9], almost all the data transferring tasks using QRStream are successfully finished.

C. *IMPLEMENTATION*

To prove the availability of QRStream, we have implemented a prototype library of QRStream and also provided a demo online basing on the library.

1) QRStream Libraries: We have implemented two prototype libraries in Javascript for Data Encoder and Stream Decoder. The encoder library provides users an API to initialize QRStream with parameters, and then the users can load the content to QRStream object and let it encode the content to the QR code stream. While in the decoder library, it uses the captured QR code frames as input, and decode them to slices of data, and then merge the data according to the sequence number.

2) Parameters: Except the parameters introduced in Section II-A, QRStream uses three other important parameters in the implementation phase which are qr code size, qr code capacity and time interval of frames. All of these parameters will have an effect on the capture speed: if the width is too small, or the qr code capacity is too high, or time interval is too low, all of these situations may cause a low recognition speed. However, if too low qr code capacity or too high time interval, the content to be transferred in a given time will be reduces, which may also cause a low speed. For these reasons, the developer may have to tune these parameters according to their situations. In our usage, we commonly set the width to 500 px, qr code capacity to 500 bits, and time interval to 0.5 seconds.

3) Project on Github: We have made the QRStream as an open source project, and the code can be found on Github at `https://github.com/qrstream`.

III. RESULTS

This section evaluates QRStream with several experiments. We evaluate them with the client on an iPhone XR, and the server side is hosted on an Intel® NUC Kit NUC6i7KYK who has 16G memory and a 6th generation Intel® Core i7-6770HQ processor. We synthesize the workload by randomly generating text content with sizes from `1K, 2K, 4K, 8K, 16K` and `32K` which are common in practical usage.

For our evaluation, the most intuitive metric is the data transferring time consumption of the tasks, however, as there are manual operations in the data transferring steps, the time consumption is affected by the manual operation speed. For this reason, in order to make the evaluation more objective, we use the successfully recognized ratio of the QR stream in one loop as the performance metric. With this ratio, we can calculate the overall time consumption if the manual operation speed is given.

We evaluate QRStream as follows. First, we briefly evaluate the QRStream performance with different file sizes, and then select a typical file size to go on evaluation with different QR code frame intervals. Next, we evaluate the performance with different QR code capacity settings to find the effect of capacity on performance. Finally, we will select a set of potential best parameters to evaluate QRStream with different file sizes to find how good could QRStream be on text healthcare data transferring.

A. Frame intervals

In this section, we evaluate the effect of file sizes and QR code frame intervals with the synthesized workloads. First, we briefly evaluate the performance of QRStream for different file sizes. We find that when the file size is small the success ratio is not very stable, but when the file is large, the success ratio is almost stable at some value. The reason is that when file is small, it is split into a small number of QR code frames, so even only one extraction failure will lower down the success ratio a lot. So the success ratio is not stable when file is small and becomes more and more stable when the files become larger. From the experiments results, we believe that there is no strong relativity between the success ratio of QRStream and the files being transferred. For this reason, to simplify the evaluation, we select a typical file size `8K` (and the capacity is set to 500 bytes) to evaluate QRStream with different frame intervals (125ms / 250ms / 500 ms / 1 sec / 2 sec). Each test has been done 4 times.

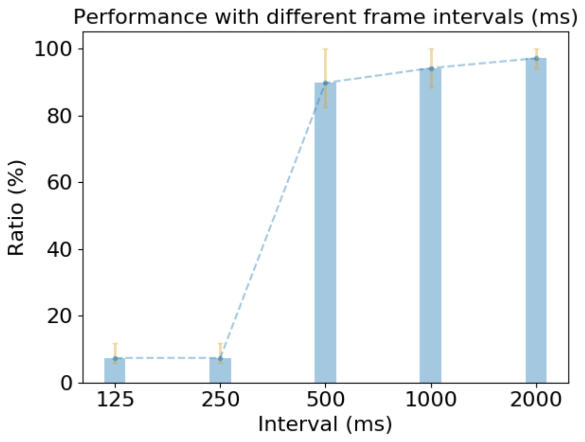

Fig. 3. Success ratio with different frame intervals (ms).

Figure 3 shows the experiment results, where we can find that the success ratio becomes large according to the frame interval when the frame interval is less than 500ms, and become stable when the interval is larger than this threshold. This is because of that, when the interval is smaller than the the QR code extraction time, it will fail frequently. And when the interval becomes larger, QRStream will have enough time to extract information from the captured QR code frame, and the success ratio will be relatively high. So, to use QRStream efficiently, we should set the frame interval to a value which is a little larger (but not much larger) than the QR code extraction time (which is related to the performance of the mobile device). In our experiment settings, the threshold value is 500ms.

B. QR code capacity

In this section we mainly focus on evaluating the relationship between the success ratio and the QR code capacity. We use different QR code capacities (including 256 / 512 / 1024 / 1536 bytes) to transfer a random generated 10K file, and evaluate the success ratios of the QR code extraction by QRStream to find the relationship.

Figure 4 shows the experiment results, where we can find that the extraction success ratio is high when the capacity

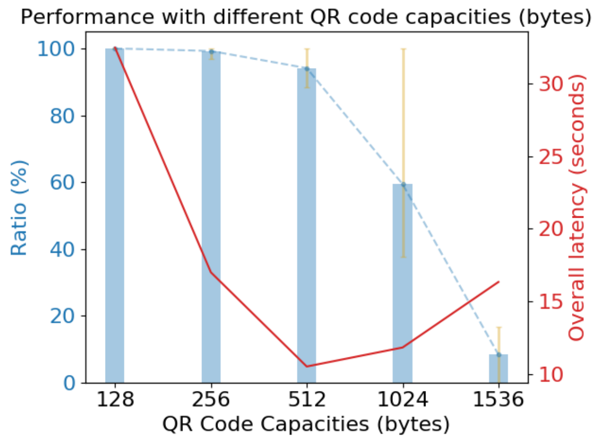

Fig. 4. Success ratio with different QR code capacities (bytes).

value is small, and becomes lower and lower when the capacity value is larger than 512 bytes. This is mainly because of that when the capacity is low, the QR code is easy for mobile devices to extract them out. When the capacity becomes large, the QR code frame become complex to be quickly extracted out. Actually, the QR code capacity affects the transferring performance in two aspects which are the number of QR code frames and the QR code extracting time. The overall time consumption can be approximately calculated by the following formula:

$$(\frac{Size_{content}}{Capacity}+1)\times Interval+Count_{failed}\times Time_{manual}$$

, where $Size_{content}$ is the content size to be transferred, $Capacity$ is the QR code capacity, $Interval$ is the QR code frame interval in the stream, $Count_{failed}$ is the failed ones which are not been recognized in the first capturing loop, and $Time_{manual}$ is the operation time to put the missing frame ID which actually is an objective parameter related to the person who use it. In our experiments, we always do it in about 1.5 seconds to input the missing ones, so the $Time_{manual} = 1.5$. The overall time consumption curve can also be found in figure 4. From the curve, we can find that, if the capacity is set to a small value, then there will be more QR code frames to transfer but easier to be extracted for each frame, otherwise if the capacity is set to a large value, then there will be less QR code frames but harder to be recognized. So the user of QRStream system should select a balance value for the QR code capacity for a better overall performance. In our settings, we can find that the 512 bytes is the best option for the QR code capacity parameter, and the overall latency is about 10.5 seconds.

C. Overall performance

With the experiments in the previsous subsections, we found that setting file size to 8K bytes, frame interval to 500 ms, and QR code capacity to 512 bytes would be a good choice for QRStream in our experiment environment. Thus we set the parameters to these values, and take experiments on the overall performance of QRStream. We did 4 times on transferring a 8K bytes file through QRStream, and the result shows that we can transfer a 8K bytes size file in about 9.2 seconds in average. With this performance, our proposed method can keep the health data delivery latency at a small value which is at least as low as, and commonly much lower than, the traditional paper based report delivery method where data is always transferred in tens of seconds.

IV. DISCUSSION

In order to explain the QRStream system more detail, in this chapter, we will first show two important potential scenarios, followed by analyzing the advantages and limitations of QRStream.

A. Potential Scenario

QRStream is designed to be used in transferring health data for healthcare organizations and users. It can be used in many different scenarios. **Clinical Data Delivering in Hospital** is one of these scenarios. Traditionally, after a patient goes to hospital for a clinic activity, he or she will get their clinic result report printed on paper. But it is not good for delivering, sharing, and management. While with QRStream, we can make it much more convenient. In order to add the QRStream function to the traditional systems, the hospital only needs to add a new QRStream button in the traditional result print page and a new pop up page. When the patient click the button a new QRStream page will pop up. The page contains a metadata QR code which is composed by the metadata (like size, md5sum, and so on) of the data being transferred, and a stream of QR code frames which are encoded from data slices, and a component for showing the QR code of the missing data slices. And in the user device side, the patient will first capture the metadata frame to extract the meta information of the data being transferred, after that the QRStream client will guide the user to go on capturing the QR code stream. For the impact of the environment, for example dim light, the client might not be able to capture all the stream frames, and some frames of the QR code stream may missing, then the users can input the missing IDs, which are showed in the client App, into the missing frame ID on the terminal of the hospital, and capture the showed QR code on the terminal. With this workflow, the user clinical data will conveniently pop up to the user device from the terminal of the hospital. All the functions of metadata and stream QR code frames generating are implemented in the QRStream library which is available on the GitHub repository. The repository also contains a demo system for the hospital scenario. You may refer to the demo page at `https://qrstream.github.io/demos/his.html`. With QRStream, we can meet the requirements of most scenarios where we need to transfer text health data.

B. Advantages

As we described, QRStream uses QR code streaming as the transferring media and uses camera at client side to capture data without any physical connection. In this way, the QRStream system has the following advantages.

Same security level as traditional method. In traditional paper printing based record delivery method, the HIS system always uses three steps to send the record to the users which are collecting information by the system, rendering content in some form like table, and then printing on paper for the users to take out. Similarly, QRStream shares the same steps and same connection requirement as the traditional method. So the HIS systems are at a same security level which is secure and no potential data leaking or viral infection risks exist.

Easy to be integrated into healthcare systems. To integrate QRStream with existing healthcare related system, the developers only need to load our QRStream library to the project, and feed data to an provided API and display generated QR code frames in their own way. While in the client side the developers can capture the QR code streaming frames with their favorite tools, and feed the captured frames to our client side API, and the original data will be reproduced. Besides, the example code and demo page code both for client and server side are open source project, and the developers can implement their own system basing on our demo code.

Convenient for text health data. Traditionally, to deliver the digital version of the health data to the users, the operators from the healthcare organizations may copy user's record to a flash driver or burn the data to a CD / DVD, which may take a long time to be finished or even cost the users more money. Comparing to these methods, QRStream performs much more convenient and faster. For healthcare records, most of them can be transferred in a small latency, for example, less than 10 seconds per record. And only a device with a camera, like mobile phone, is needed to transfer the data. Users can conveniently click a button on the server side and capture the data through the client and take their own data out.

C. Limitations

QRStream is convenient for text health data transmission, however, it has several limitations.

Data size. The amount of data that can be stored in the QR code symbol depends on the datatype, version, and error correction level [9]. For example, the maximum storage capacity for binary data with version 40 and error correction level L (low) is 2,953 bytes. And in practice, one QR code frame may contain only 1,000 bytes or even less for better extraction performance. In this way, to transfer large data with QRStream would need to play a large number of QR code frames, for example, 1000 frames for 1 Megabyte data. If we set the play speed to 2 frames per second, it would take 500 seconds to finish the data transmission. It is not an acceptable latency. For this reason, QRStream is aimed at transferring the small text health data.

Data type. Text data is one of the most common health data types, however, health data might be some other binary types like image, video or audio data. And those binary data types are always come with large data size. But as we stated that QRStream is limited for small size data, it is not suitable to transfer those large data in those binary types.

However, in practice, most of the generated data (not in size) are text health data, like clinic data, examination data or the test data and so on. Although QRStream is limited to be used in transferring small text health data, it still can meet most of the healthcare data collection and transmission requirements.

V. CONCLUSIONS

In this paper, we propose QRStream, a secure and convenient method for transferring text healthcare data from the organizations, like hospitals, to the patient users when a new record is generated. For the organizations, it provides a method to deliver the health data without any physical connection to the users, which makes sure that the machines will not be infected or hacked and no data leak will happen because of the user data sharing. For users, it makes sure that the users can conveniently obtain their own health data from the organizations in time for their further utilization. The evaluation shows that QRStream can transfer text health data smoothly with an acceptable and useable performance, for example, transferring 10K data in 10 seconds.

QRStream is an open source project available at `https://github.com/qrstream`, and the homepage is `https://qrstream.github.io`.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their comments and kindly suggestions.

REFERENCES

[1] C. Safran, M. Bloomrosen, W. E. Hammond, S. Labkoff, S. Markel-Fox, P. C. Tang, and D. E. Detmer, "Toward a national framework for the secondary use of health data: an american medical informatics association white paper," *Journal of the American Medical Informatics Association*, vol. 14, no. 1, pp. 1–9, 2007.

[2] G. Eysenbach, "Medicine 2.0: social networking, collaboration, participation, apomediation, and openness," *Journal of medical Internet research*, vol. 10, no. 3, 2008.

[3] P. Kostkova, H. Brewer, S. de Lusignan, E. Fottrell, B. Goldacre, G. Hart, P. Koczan, P. Knight, C. Marsolier, R. A. McKendry, *et al.*, "Who owns the data? open data for healthcare," *Frontiers in public health*, vol. 4, p. 7, 2016.

[4] C. J. Haug, "From patient to patient—sharing the data from clinical trials," *New England Journal of Medicine*, vol. 374, no. 25, pp. 2409–2411, 2016.

[5] N. Genes, S. Violante, C. Cetrangol, L. Rogers, E. E. Schadt, and Y.-F. Y. Chan, "From smartphone to ehr: a case report on integrating patient-generated health data," *npj Digital Medicine*, vol. 1, no. 1, p. 23, 2018.

[6] A. J. Greenberg, D. Haney, K. D. Blake, R. P. Moser, and B. W. Hesse, "Differences in access to and use of electronic personal health information between rural and urban residents in the united states," *The Journal of Rural Health*, vol. 34, pp. s30–s38, 2018.

[7] M. Meingast, T. Roosta, and S. Sastry, "Security and privacy issues with health care information technology," in *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*. IEEE, 2006, pp. 5453–5458.

[8] J. J. Cimino, V. L. Patel, and A. W. Kushniruk, "The patient clinical information system (patcis): technical solutions for and experience with giving patients access to their electronic medical records," *International journal of medical informatics*, vol. 68, no. 1-3, pp. 113–127, 2002.

[9] T. J. Soon, "Qr code," *Synthesis Journal*, vol. 2008, pp. 59–78, 2008.