Media in Credentials

draft	Current state of this PoA
	Possible states
	• brainstorm: still in infancy; major changes likely
	• draft: getting reviewed; changes likely
	• approved: only anticipate minor changes
	  ○ changes accompanied by a call-out-comment
	  ○ typos don't need comment

Daniel H	Responsible

Note: Previous POC done by sklump at BCGov is here, here, and here; it resulted in a demo feature but not something robust yet. The brief summary for current work is that we need a way for VCs to include (either directly, or by reference) data other than strings, numbers, and dates. For example, we need a VC to include a photo or similar biometric template of its holder, a PDF of a hand-written and hand-signed doc, an audio recording, a video, a genome, etc.

Assumptions

Privacy

Photos, genomes, PDFs, and other large BLOBs are strongly identifying because they are globally unique. Defeating correlation while sharing them is conceivable, but expensive and fraught with complication. Therefore we assume that A) the initial versions of this feature are NOT privacy-preserving confidence=high; B) this will be acceptable to the marketplace confidence=medium; and B) the user will be informed/coached appropriately; confidence=medium. Media in a credential should be thought of exactly like social security or passport numbers in a credential: there are reasons to encode such data and share it, but the user should be warned that doing so will eliminate their ability to remain uncorrelated. → UX requirement in C.M

HUB Maturity

DIF Hubs and similar tech are very early in the standardization arc--both as specs and as implementations. Therefore, although we could build a feature that stores the media from credentials in a hub on issuance, and that references the hub location during presentation, the feature would be built on a shaky foundation. We assume that hubs will mature to the point where we can build a feature that uses them, sometime in late 2020 or beyond. confidence=high, risk=low.

Caveats Okay

The first delivery of the feature will be suitable for production use cases, but will have some scale and efficiency caveats. Specifically:

- We are willing to tolerate a modest amount of bloat from repeated base64-encoding of binary data during credential and proof exchange. This bloat might, for example, turn 1MB into 3MB--but it won't turn 1 MB into 100MB.
- Storing large numbers of credentials with large amounts of media in them will strain a mobile wallet's storage subsystem, so we don't expect customers to do this, as a general rule.
- Individual credentials with massive amounts of media in them (say, more than a dozen or so items) may also strain the display UX in C.M, and/or storage and transmission mechanisms. We don't expect customers to do this often.
- Individual media items of an unreasonable size (say, 100MB) may make handling routines in various places run in a very suboptimal way. However, the routines shouldn't crash, and they should eventually succeed.

Tests

1. Acme (running Verity) can issue a credential to Bob (running Connect.Me), where a field of the credential is a JPEG of size=~500 KB. Bob can receive the issued credential and store it successfully. He can also display the credential; the field associated with the image either displays an image directly in C.M, or displays a clickable action that displays the image. The size of the message on the wire is between 500 KB and 1 MB. Bob's local storage grows by between 500 KB and 1 MB as a result of accepting the credential.
2. Same scenario as #1, except that the issuer sends an incorrect attachment (the photo doesn't match its hash from the credential). The holder (Bob using C.M) emits a problem-report message describing the problem accurately.
3. When Bob backs up his wallet after receiving the credential from test 1, the wallet backup contains the image, as demonstrated by the fact that if Bob deletes the stored image from disk and then restores the backup, the image is back on disk.
4. C-corp (running acapy that doesn't support this feature) can challenge Bob to prove something with his credential, where the proof request does NOT include anything related to the image field, and a proof presentation generated by Bob with C.M is considered valid by C-corp.
5. Acme (running Verity) can challenge Bob to prove something with his credential, where the proof request DOES include a disclosure of the image field, and the presentation generated by Bob with C.M will validate with Verity.

6. Acme (running Verity) can challenge Bob to prove something with his credential, where the proof request DOES include a disclosure of the image field, and the presentation generated by Bob with C.M will validate with Verity.
7. Same scenario as #6, except that the presentation includes an incorrect attachment (the disclosed BLOB hash doesn't match the attached photo). This should cause the verifier to reject the proof.
8. Same scenario as #1, except that the attachment is only 1000 bytes with mime type application/pdf, and other sizes are adjusted accordingly. All behaviors and outcomes should be identical, except that C.M should NOT display the PDF inline, but rather offer to launch the default handler for PDFs.
9. Same scenario as #1, except that the attachment is a 100MB .mp4 file, and other sizes are adjusted accordingly.
10. Same scenario as #1, except that the image now includes 3 small BLOB fields (~25 KB each) of various MIME types.

Phases

Work to deliver this feature will occur in phases, because the ecosystem isn't mature enough--and we are not mature enough, either--to support the fancy incarnations of this feature that will eventually be possible. Also, the later phases are much more expensive, and they involve more interoperability risk than we should incur right now.

Phase 1 (required)

This encompasses work that we could do in Q1 of 2020 with relatively low risk and only modest effort. The deliverable will be a feature where any media can be attached to an issued credential, and can be shared with a verifier as an attachment as well. This feature will be releasable in production, but it will not be suitable for heavy use because the existing KMS ("wallet") in Indy SDK stores credentials in a way that's inefficient. Reasonable use will be a holder that has a handful of credentials with a small number of "media fields." Unreasonable use will be a holder that has tens of thousands of credentials with associated media; this will bloat the wallet and make its backup problematic. We expect that phase 1 deliverables will be useful throughout 2020 and beyond; even when later phases are delivered, phase 1 mechanisms will remain a viable option.

Only phase 1 is required to have a production-usable feature; however, phase 2 would also be a wise investment since it fixes some scaling issues.

Phase 2 (recommended)

In this phase, credential storage inside the KMS is changed, making it practical to hold far more credentials that contain media. Also, the issuer, holder, and verifier codebases are upgraded to support indirection--instead of directly transferring media during issuance or verification, only a

hyperlink to media is transferred, and downloads of the media happen separately. This lays the foundation for credential media to be stored in a general bitbucket like imgbin, Google Drive, DropBox, etc.

Phase 3 (optional)

This is a logical variant of phase 2, where storage is to a personal data hub as conceived by DIF (or W3C's encrypted data vault initiative, etc). Today, the design of hubs is not mature enough to use them. If it is mature enough by the time we begin phase 2, perhaps we can collapse phase 2 and phase 3. If not, then we can tackle them separately.

Phase 4 (optional)

This adds privacy features. It may begin in parallel with phase 2 or phase 3, but is unlikely to be a sellable feature in the same timeframe. See the Privacy section for more details.

Phase 1 Approach

Document and code support for a convention for BLOB handling in credentials. (EV must raise a community PR against RFCs 0036 and 0037 describing these conventions.) This includes:

- Fields that convey attestations from the issuer about BLOBs follow the naming convention `<descriptiveName>_hashlink`. Such fields are called ***hashed BLOB fields*** (***HBFs***). HBFs contain a hashlink[1] to their content, rather than the content itself.
- Issuers that build credentials like this should include the raw BLOB for each HBF, as an attachment to the `issue-credential` message, at the end of the issuance protocol described in RFC 0036. This means the holder will receive the bytes of each BLOB associated with an HBF, as part of receiving the credential itself.
- During verification (RFC 0037), a proof should include the raw BLOB for each HBF that's disclosed, as an attachment to the `presentation` message. This means the verifier will receive a hash of the BLOB as a disclosed field value, plus the raw bytes of the BLOB as an attachment, and can compare the two to ensure correctness.
- The verification procedure coded in indy-sdk must be updated to account for this convention, such that a verification of a proof involving HBF fields compares the hash against the bytes of a BLOB.
- C.M should be updated so it displays HBF credentials as if the BLOBs were inlined (the UX is that the user feels like the credential contains their photo or PDF). It should also warn the user about the privacy consequences of disclosing such BLOBs, and it should generate the correct proof for a proof request request that asks for an HBF to be disclosed.

[1] Hashlinks are currently described in a somewhat immature draft IETF RFC. They are basically URLs that include a hash of the content so the content can be recognized as unmodified, even if downloaded from a different place. See https://tools.ietf.org/html/draft-sporny-hashlink-04.

Separate from the convention, but part of our phase 1 delivery, is C.M's storage of credentials that have associated BLOBs. It would be preferable to store these BLOBs outside the KMS, and to update the KMS such that BLOBs are incorporated in the KMS only by reference (KMS stores/indexes a path to where they are stored on disk, never the actual bytes). However, if we must store the BLOBs directly in the KMS in phase 1, that is an acceptable fallback position.

The answer to the need in the previous paragraph (where to store the BLOBs) will affect KSM (wallet) backup. If we store the BLOBs inside the KMS, then the KMS backup will automatically include BLOBs. This imposes almost no new cost on us, but it makes backups problematic because they will quickly become large, and it won't be possible to treat BLOBs separate from private key data for backup purposes. If, on the other hand, we modify the storage of BLOBs so we only place a *path* in the KMS, but not the BLOB itself, then we have to add a feature to our wallet backup where content in paths outside the KMS are also part of the backup scope by default. Possibly we could add a feature to make this optional/selectable, in a later phase.

Also separate is the way C.M's UX will be updated to display credentials using this mechanism. The assumption is that images (and possibly video?) would be displayed inline with other attributes, but that other types of BLOBs (e.g., PDFs, biometrics) would be displayed as a link to a file that invokes its default handler on the platform.

Staged Delivery

- Draft an update to RFCs 0036 and 0037, describing the above conventions, and share via PR with the community. Milestone: RFC Draft.
- Modify Verity so its issuing code understands and automatically uses these conventions (hashing BLOB as the HBF field content; attaching the BLOB to the `issue-credential` message) anytime it sees a field in a schema named *_hashlink.
- Modify the Indy/Aries KMS so it supports storing BLOBs--preferably by dumping them in a folder outside the KMS, and storing only a path, but optionally just storying the bytes of the BLOBs in the KMS itself (non-secrets API).
- Modify C.M so it displays HBF creds correctly by looking up the MIME type of the HBF field and invoking a default handler for that MIME type. (Images may be specially handled inline in C.M's display, whereas PDFs and other types may load a different app or a browser?)

Appendix: Privacy-respecting BLOBs

If a BLOB is simply disclosed, it becomes a strong correlator. To prevent correlation, the following techniques are possible:

- Verifiable computation (e.g. ZK-SNARKs) can be used to permute what's disclosed. The permutation algorithm will be different for different types of data (e.g., changing a pixel in an image requires one algorithm; slightly shifting an attribute in a fingerprint requires a a different algorithm; changing a harmless header in a .PDF requires yet another algorithm). This proof is expensive to generate and verify. We've done a POC of it, and it does work in concept, but it seems like an expensive feature to implement, and it wouldn't be usable with mime types that don't have an implemented permutation algorithm.
- Multiple references to what's logically the same BLOB, pre-permuted by the issuer, could be embedded in the same credential (e.g., 100 variants of the same mug shot). The prover could choose one at random (or very deliberately) to break correlation.
- The disclosure could be to a randomly chosen third party that agrees to permute it before passing it along. The permuter could be crowd-sourced. There would be no incentive to cheat, but also no need to do verifiable computation.
- With BLOBs that are biometrics, multiple low-fidelity biometrics can be embedded, instead of or in addition to a single high-fidelity value. This allows progressively greater confidence with progressively greater disclosure. This technique is described in more detail here.
- Also with BLOBs that are biometrics, proof of biometric match can be done by a different party from proof of match of all other attributes. The biometric service provider (BSP) sees only the biometric data, never anything else--and the normal verifier sees only the other data, never the biometric. This diffuse trust may be better than nothing, although the BSP and the verifier could collude. This is discussed in more detail here.