FIPS Compliance and Hyperledger Ursa/Indy/Aries

There are common questions about Federal Information Processing Standards (FIPS) compliance regarding Hyperledger Ursa, Hyperledger Indy, and Hyperledger Aries. First, it is important to understand that there is a difference between FIPS compliance and FIPS validation.

FIPS compliance has multiple levels, and the majority of entities and products are actually not FIPS compliant. FIPS Level 0, as I call it, means using approved algorithms as defined in standards like FIPS 186-5—a standard for digital signatures and encryption. That's where most organizations stop. They just use the approved specifications and move on. There is more to it than that. The next step is using FIPS-validated software or hardware. For example, OpenSSL has been FIPS validated, meaning that government or financial organizations are approved to use this software for protecting their data. Usually this is for FIPS 140-2, the Federal Information Processing Standard for hardware, software, and firmware solutions to ensure end users receive a high degree of security, assurance, and dependability.

FIPS Level 2 requires adding physical and role-based security requirements to detect tampering. Some cloud providers and virtual HSMs implement this, such as Unbound, AWS, and Azure.

FIPS Level 3 requires adding identity-based authentication and physical mechanisms for tamper detection and tamper response procedures, such as erasing keys from EEPROM if the system is illegally changed.

FIPS Level 4 adds environmental tampering detection. This includes things like voltage- and heat-attack mitigation, among other things.

So, when someone asks, "Are you FIPS compliant?", this means you are using only FIPS-validated hardware and software. This level of compliance depends on how strict end-client requirements are. FIPS Level 2 is usually sufficient for contracts with the United States Department of Homeland Security (DHS) Directorate of Science and Technology (DS&T). This means that the software libraries and hardware you use must have been validated by one of the 13 National Institute of Standards and Technology (NIST) laboratories. Ursa/Indy/Aries have not been validated by a NIST laboratory.

Now let's describe the crypto provided and used by Ursa. Ursa Anoncreds 1.0 uses OpenSSL to implement CL-RSA signatures. CL signatures are not FIPS approved and are unlikely to be, given there are better options that are more performant and size efficient. Anoncreds 1.0 also uses CKS ECC-based signatures to implement cryptographic accumulators for Revocation. CKS requires the use of elliptic curve pairings which are also not yet FIPS approved. Other crypto primitives that Ursa offers are:

Signatures: Ed25519, Ecdsa-secp256k1, BLS

Encryption: AES-GCM, AES-CBC, XCHACHA20-POLY1305

Key agreements: Ecdh-secp256k1, x25519

Hashes: SHA2, SHA3, Blake2

You may recognize some of these algorithms are FIPS approved, but Ursa as a library is not FIPS-validated.

Anoncreds 2.0 is very different from Anoncreds 1.0. Anoncreds 2.0 uses ECC-pairing-based signatures and proofs. The verifiable credential signatures offered are BBS+, Pointcheval-Saunders, and Groth. These support zero-knowledge proofs for the attributes and signatures. Ursa also has implementations for Bulletproofs. All of these primitives currently use BLS12-381 as the underlying curve, which is also not yet FIPS-approved. However, the code is written so that any pairing-friendly curve can be used. As mentioned previously, no pairing-friendly curve is FIPS approved yet. BN curves are more likely to become FIPS approved given they are already used in production Trusted Platform Modules (TPM). If any pairing-friendly curve becomes FIPS approved, Ursa might switch and use that curve instead. The reason for using BLS12-381 instead of BN curves is BN curves require higher bit-strengths like 512 bits vs 381 bits and are susceptible to more attacks. Higher bits impact performance and space. Anoncreds 2.0 uses Merkle-Proof-based revocation instead of the CKS signatures because Merkle-Proofs are smaller and faster. Anoncreds 1.0 required two credentials when using revocation, one under CL and one under CKS. Anoncreds 2.0 removes this requirement.

Next is the cryptography used in Indy. Indy uses ZMQ for node-to-node, node-to-client communication. ZMQ uses Ed25519/x25519 for session management and authentication based on libsodium. The libzmq and libsodium libraries are not FIPS validated. Blockchains in general use threshold signatures to reach consensus. No threshold signature is FIPS approved, so if you use a blockchain at all, you have now stepped outside the FIPS "kingdom." Basic signatures could be used in blockchains like ECDSA but at a huge performance and space cost. Blockchains are already slow, and most choose not to incur this penalty for the sake of being FIPS compliant. DHS is still excited about blockchains, so they might overlook this requirement for the time being. Indy uses BLS signatures based on curve BN-254 for consensus. The Merkle tree uses SHA256. Indy uses Ed25519 for transaction author validation, i.e. transactions signatures submitted to be recorded must be Ed25519. Indy uses libsodium for Ed25519 signature validation. Transaction authors can use whatever Ed25519 implementation they want.

Aries implementations have a wide variety of implementation languages from GoLang, C#, Java, nodeJS, and Rust. None of them are using FIPS validated libraries. NodeJS can use OpenSSL under the hood, but it depends on what the implementers intend to do. C# on Windows can use FIPS 140-2 approved algorithms, but it’s not strictly enforced.

As far as roadmaps go, the Ursa community is advocating to NIST to approve the algorithms in Ursa first. Ursa does not currently have plans to be FIPS validated because this is a long and costly process, and until the algorithms are approved, it is ineligible to be FIPS validated.