No call authentication. When securing a conversation, *confidentiality* (hiding the contents of the conversation) is not the only desirable property. Without also *authenticating* the conversation, an attacker located in the network (e.g., a malicious actor intercepting traffic on a user's LAN, or a compromised ISP) can insert themselves between the endpoints. Wire provides both properties for text-based conversations.

The Wire protocol uses SRTP to secure audio/video calls. SRTP requires an external mechanism to establish the initial cryptographic key to protect the data. Wire uses a DTLS handshake to establish this initial key. DTLS is a UDP-based analog of TLS (used to secure HTTPS web browsing). Consequently, the key exchange in a DTLS handshake needs to be authenticated by a certificate authority, as in the case of web certificates. Based on some quick packet captures, it appears that the certificates used by Wire for this DTLS handshake are self-signed. If this is truly the case, then the handshakes do not provide any authentication in the presence of an active network attacker (i.e., a network attacker that can modify traffic), making Wire audio/video calls vulnerable to man-in-the-middle attacks.

The Wire security whitepaper says that "the authenticity of the clients is also verified during the handshake", but this is not consistent with the use of self-signed DTLS certificates. If the certificates are authenticated by the central Wire servers, then this prevents call interception by anyone *except* for the Wire operators.

In either case, the use of DTLS here is highly unusual and the security it provides is sub-optimal. A better approach would be to use a protocol like ZRTP (the standard used by Signal, Jitsi, Linphone, and others) to establish the initial SRTP key, or perhaps a more advanced key exchange scheme. The authentication for this scheme should *not* be based on certificates (as in DTLS); instead, the authentication should be bound to the long-term keys that are used by Wire's text-based conversations. A very simple solution would be to transparently perform a well-known unauthenticated key exchange (e.g., Diffie-Hellman) over an authenticated Wire text conversation, and to use the resulting key to initiate SRTP for the audio/video call.

The Wire security whitepaper mentions that authenticating audio/video calls against the cryptographic identities used for the text messaging protocol is future work, so this issue appears to be known.

Answer
Calls are authenticated. This is done by the Wire servers and mitigates MitM attacks on the network layer.
As you pointed out correctly the authentication is not done end-to-end. This is a known limitation mentioned in the whitepaper. It is also something that is about to change.

Here is a bit of background information on how we took certain decisions: One of the design goals for Wire was to also work in a browser. For any kind of calling browsers use WebRTC, which then also became the natural choice for the native Wire clients. WebRTC is a well-tested and well-documented framework and is indeed based on DTLS-SRTP. ZRTP never became a WebRTC standard, which is why we didn’t consider it as an option initially.

When a call is set up, the DTLS part requires what is called “self signed certificates” in the WebRTC terminology. In short you can think of it as a keypair that is generated by the browser (and that you don’t really control as a developer). You do get to see the public key, and you have to transmit that key to the other side (you choose how, this is outside of the scope of WebRTC). Conversely you also receive the public key of the other side and you feed it into the DTLS handshake. This makes it easy to verify you speak with the correct partner, but it all comes down to how the keys are transmitted. In order for this to be end-to-end secure, the channel in which the keys are transmitted needs to be secure.
When Wire launched initially, text messages were not end-to-end encrypted, therefore no E2EE channel existed. When it was added earlier this year to replace the old mechanism, it became the natural choice to be also used to verify the call authenticity. We therefore set out to design a new signalling protocol that makes full use of the end-to-end encryption. Designing such a protocol that also supports multi-device scenarios and group calls turned out to be non-trivial. While end-to-end encryption is a desired property it quite often also is the natural enemy of robustness and efficient real-time synchronisation.
To make a long story short, we will now launch the first version of our new end-to-end encrypted signalling protocol.

TL;DR: Calls are authenticated, but not end-to-end authenticated. It is a known limitation that will be addressed in the following weeks.

Questionable codecs: It has been known for some time (e.g., WMM06, WBMM07, and WMSM11) within the system security academic community that using variable-bitrate codecs within an encrypted tunnel leaks information about the communication. Since encryption does not hide the *length* of the messages being transmitted, the bitrate of a conversation over time can be observed by a passive network attacker, even without access to the encryption keys in use. In some cases, knowledge of the bitrate changes is sufficient to reconstruct the unencrypted conversation with high accuracy. Consequently, while variable-bitrate codecs use the available bandwidth more efficiently, using constant-bitrate codecs is important for security.

The security whitepaper mentions that Wire uses the Opus codec to transmit audio data. The whitepaper does not mention what codec is used for video data. Opus supports both variable- and constant-bitrate encodings, so it is unclear if Wire is vulnerable to traffic analysis. In any case, Wire should ensure that constant-bitrate codecs are used for both audio and video data, and that this is clearly stated in the security whitepaper.

Answer
We addressed that question previously:
https://medium.com/@Be.ing/does-the-bitrate-vary-with-the-input-signal-or-only-the-network-conditions-c9bb16593396

While it is true that VBR codecs can theoretically leak some data, it is very questionable whether that data has enough entropy to be useful at all.
The biggest problem is obviously for pre-recorded audio data, not streamed audio data. Wire only uses Opus VBR for streaming.
The motivation behind using VBR rather than CBR is simply that VBR uses less bandwidth, meaning that calls will either sound better, or can actually work on very slow networks. Enforcing the use of CBR will clearly degrade the perceived and measured call quality.

We will keep monitoring the research around that subject and we are also considering introducing an option to switch between VBR and CBR.

Web-style account authentication: The Wire client authenticates with a central server in order to provide user presence information. (Wire does not attempt to hide metadata, other than the central server promising not to log very much information.) The Wire whitepapers spend an unusual amount of space discussing the engineering details of this part of the protocol. However, the method of authentication is the same as it is on the web: the Wire client sends the unencrypted, unhashed password to the central server over TLS, the server hashes the plaintext password with scrypt, and the hash is compared to the hash stored by the server. This process leaks the user's password to the central server; the server operators (or anyone who compromises the server) could log all of the plaintext passwords as users authenticate.

Wire likely designed the protocol this way in order to make it easier to support web-based clients. Authentication mechanisms like this are standard on the web for historical reasons. However, since Wire has complete control over both the client software and the protocol, it is possible to do better. Wire should use a password-authenticated key agreement or a more sophisticated challenge/response protocol for user authentication. The advantage of these schemes is that they authenticate users *without* making the passwords vulnerable to data breaches on the central server. While the system would still be vulnerable to attackers capable of releasing malicious Wire updates, this would be a significant improvement.

Answer
This is a valid point. As mentioned earlier, Wire was initially designed to be fully compatible with browsers and this is where this form of authentication comes from. That doesn’t mean it can’t be improved, and we have already looked at alternatives.

Code structure and complexity: The Wire implementation is extremely complex, making the attack surface relatively large. The desktop application is implemented as a packaged web application. Each time that the application is launched, large parts of the code (written in JavaScript) are re-downloaded and executed from the Wire servers without user interaction. Essentially, the Wire app itself is merely an embedded web browser. There does not appear to be any notion of "JavaScript pinning".

A common weakness of all secure messaging applications is that the application must be downloaded, trusted, and updated regularly; if the update mechanism is ever compromised, the system can be made to arbitrarily deviate from the specifications to the benefit of the attacker. However, the current design of the Wire desktop application makes such attacks simple, difficult to detect, and rapid. Moreover, the complexity of the application and its libraries presents many opportunities for security vulnerabilities. While this complexity may be necessary to support the rich feature integrations of the platform and the broad system support, the cost is more opportunity for exploitation.

Answer
The most important part is certainly the encryption library -- Proteus -- which exists in only two implementations for all clients: Rust and JavaScript and it has been audited a few times. This greatly reduces the complexity you mention.
As you correctly point out, the problem you mention is applicable to any kind of software you download from a server. This is also one of the reasons why Wire clients are open source and you can build your own client without having to rely on any server. We are generally aware of this weakness, and we are exploring improvements.

Closed-source server: The Wire client is open source—and open to analysis by security researchers—but the server software is not.

The problems listed above weaken the security of Wire relative to competitors like Signal, but the problems are not insurmountable. The chat features offered by Wire have a very modern aesthetic that is very popular with users, and this makes Wire a very interesting offering. When these security problems are addressed, users may want to consider using the system. In the meantime, users should avoid using Wire audio/video calls for secure conversations, assume that Wire passwords could be silently compromised, treat the Wire application like a constantly updating web service rather than a semi-stable desktop application, and consider sandboxing Wire on sensitive systems.

Answer
We will open source the server source code in Q1 2017.
Open sourcing large codebases takes some time, and we decided the clients had to be open sourced first since they are far more important given the end-to-end encryption Wire uses.